FraTAct for Transforming A Nescient Process Activity Into an Intelligent Process Activity

Rafiqul Haque[1*] and Nenad B. Krdzavac[1]

[1]Department of Accounting, Finance, and Information System, College of Business and Law, University College Cork, Cork City, Cork, Ireland.

Abstract

Existing business process technologies support defining only nescient activities. Currently there is no solution that underpins transforming a nescient activity into intelligent activity. In this paper, we address this shortcoming of the state of the art. We offer a framework 'FraTAct' for transforming regulation intensive nescient activities of a financial service business process into intelligent activities.

Financial service industries has been experiencing enormous challenges since the last decade. A recent financial crisis has unearthed various weaknesses in terms of administering the financial service industries. In order to prevent the future crisis, the regulators are constantly formulating new rules and also forcing the financial service industry to enact financial regulations in their financial service based application which automates financial operations.

A financial service application underpins the financial service business process that contains activities. A nescient activity within a financial service process is prone to the risk of producing an inconsistent outcome that results in severe legal consequences for a financial institute e.g., a bank. In order to avoid these legal consequences, a financial institute should develop their financial service processes by composing activities that should be intelligent to understand and comply with financial regulations. Intelligent activities will produce outcomes that are consistent to financial regulations. It will reduce the possibility of financial regulation noncompliance in financial service process based application.

Received on 02 March 2013; accepted on 07 November 2013; published on 04 March 2014

Keywords: Business Process, Nescient Activity, Intelligent Activity, Ontology, Description Graph

doi:10.4108/sis.1.2.e4

1. Introduction

A process simply put, is a set of ordered activities [15]. Activities carry out *operations* while a process is running and producing outcomes. The notion of process is used in different domains such as software process, chemical process and so forth. In this paper, process refers to financial service process and activities are financial service activities. In a financial service process (FSP), the operations must be performed in compliance with *financial regulations* that are essentially financial rules. An operation produces an inconsistent outcome if it does not comply with the relevant regulations. Inconsistent compliance outcomes result in severe consequences for an organization regardless of whether it is a financial or business organization. The *Enron-Scandal* [12] is a practical example. Therefore, compliance is a highly significant issue in FSP.

Activities within FSPs are *regulation-intensive* which means that they are based upon financial regulations (e.g., Basel III accords [3]). However, some activities within an FSP may not be regulation-intensive. A regulation-intensive activity within an FSP is called *nescient* if it lacks explicit knowledge about the underlying regulations, it lacks correlation with financial regulations, and the activity cannot produce consistent outcomes. A nescient activity is not able

*Corresponding author. Email: arhaque@ucc.ie

†This research paper is an extension of our research published in [10].

to perform operations that comply with financial regulations. This promotes the need for activities that are able to perform operations in compliance with regulations.

Existing business process technologies (e.g., BPEL [21]) lack the constructs for defining (modeling) intelligent activities in a financial service process. Some approaches (e.g., Business Process Model and Notation (BPMN) [22])facilitate annotating meta-data of a process activity to give the semantics of that activity. However, these approaches cannot be used for FSPs due to the complexity and ambiguity in financial regulations. More specifically, an FSP is slightly different from the classical business process activity as it must be aware of one or more financial regulations which can be a conjunctive statements. It may not be easy to annotate an FSP with a conjunctive statement which expresses explicitly the semantics of different parts of it. In addition, transforming annotated financial regulations into machine readable ones by preserving their semantics is still an unsolved issue. A solution that supports the straightforward transformation of a nescient activity into an intelligent one is strongly required for *smart* financial service processes.

In this research, we offer a framework named *FraTAct* that underpins transforming nescient activities into intelligent activities within the FSP space. It is important to mention that the transformation happens *indirectly* i.e., the framework supports the nescient activities by providing the exact meaning of regulations and also assists in performing operations complying with financial regulation. In order to do so, the FraTAct framework uses the notion of the *knowledge base*. The knowledge base is developed using an ontology approach.

The remainder of this paper is organized as follows. A motivating example is presented in section 2. The preliminaries are described in section 3. In section 4, we describe the knowledge base for financial services. Section 5 presents the FraTAct framework. We discuss implementation of the proposed solution in section 6. The related works is discussed in the subsequent section. Finally, section 8 outlines the conclusion and future work.

2. Motivating Example

This sections sets out the *Common Equity Tier 1 Capital Ratio Calculation Process* model. The process model is developed based on the description provided in [3]. In order to make the example simple, a limited number of activities have been incorporated in the process. Fig. 1 shows the process model.

The process starts with performing the 'List Common Equity Tier 1 Financial Instruments' activity that

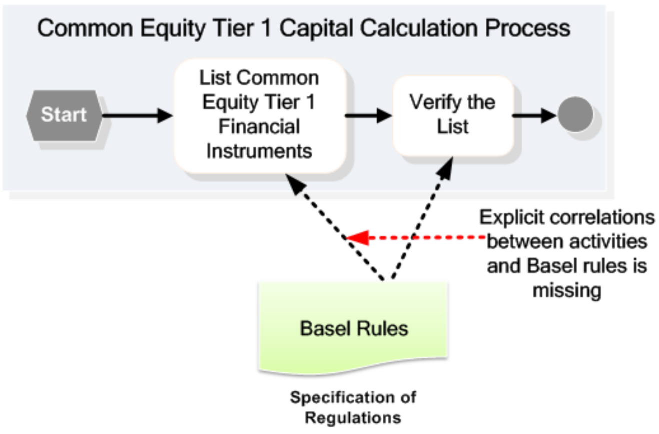

Common Equity Tier 1 Capital Calculation Process
Start
List Common Equity Tier 1 Financial Instruments
Verify the List
Explicit correlations between activities and Basel rules is missing
Basel Rules
Specification of Regulations

Figure 1. Common Equity Tier 1 Capital Ratio Calculation Process

produces a list of financial instruments. Then, the list is verified by performing *Verify the List* activity. After executing these activities successfully, the *end* event signals the *completion* of the process.

The 'Common Equity Tier 1 Capital Ratio Calculation' process is driven by regulations particularly, the Basel Capital Accords. The activities contained in this process are regulation-intensive. The operations performed by these activities must comply with the Basel accords. These accords are ambiguous. Therefore, understanding the semantics of these accords is a critical importance for the activities.

Activities contained within 'Common Equity Tier 1 Capital Ratio Calculation' are nescient as these activities lack the support of an approach that enables performing operations intelligently. The FraTAct framework supports transforming these activities into intelligent activities that carry out operations by realizing the semantics of the financial regulations derived from the Basel accords. The framework leads financial service process execution in compliance with the corresponding regulations and prevents unwanted consequences for financial institutes.

3. Preliminary

In this section, we provide a brief description of intelligent activity. In addition, this section explains ontologies and description graph.

3.1. Intelligent Activity

Intelligent activity is defined in [10] as follows:

Definition(Intelligent Activity): *Each regulation-intensive activity is an intelligent activity if and only if*

(i) it has explicit correlation with the regulations,

(ii) it relies on regulations which are encoded as axioms in a decidable description logic [13],

(iii) it cannot produce outcomes that are non-compliant to regulations,

(iv) Any activity that does not satisfy (i), (ii) and (iii) conditions will not be considered as intelligent activity within FSP.

For example, list common equity Tier 1 capital ratio calculation is a intelligent activity if it is able to interpret the semantics of the corresponding regulations (derived from the Basel Accords), complies with those regulation while executing the activities, and the activity produces the list that was desired by the users.

3.2. Ontologies and Description Graph

The Ontology Web Language (OWL2) [4] is used for modeling ontologies. The $\mathcal{SROIQ}$ DL [13] provides reasoning services for ontologies based on the tableau algorithm. The basic OWL2 concepts are classes, object properties, data-type properties, and individuals. Additionally, an ontology contains class axioms, and property assertions as well as individual assertions. Semantic Web Rule Language (SWRL) [14] is used to model rules in an ontology.

A Description Graph (DG) [18] is a directed labeled graph that contains a set of vertices, edges and a labeling function that assigns each node to an atomic concept, and each vertex to an atomic role. According to [18], a graph-extended knowledge base is a 4-tuple $\mathcal{K} = (\mathcal{T}; \mathcal{P}; \mathcal{G}; \mathcal{A})$ where $\mathcal{T}$ is a TBox, $\mathcal{P}$ is a program, $\mathcal{G}$ is a GBox, and $\mathcal{A}$ is an ABox. The $\mathcal{P}$ consists of a finite number of connected rules. The $\mathcal{G}$ contains graphs roles i.e. roles one can use in a DG [17]. Reasoning with ontology is, mostly, based on tableau [13] and hyper-tableau [18] algorithms.

4. Knowledge Base for Financial Services- A Graph-Extended Approach

As already mentioned, financial service components are built on a knowledge base (Fig. 2 shows the knowledge oriented service components). In this section, the core mechanism of knowledge base for financial services is described. Financial regulations are basic building blocks for financial services. Therefore, our aim is to develop knowledge bases for financial regulations. In this section, we explain how to encode regulation corresponding to common equity Tier 1 capital ratio calculation process (shown in Fig. 1) into a graph-extended knowledge base [18] and check satisfiability of the knowledge base using a hyper-tableau algorithm [17]. Modeling FSPs using graphs is not a new idea [28]. However, a problem arises when one needs to apply automated reasoning on the processes which must satisfy certain regulations. The execution of each activity in the common equity Tier 1 capital ratio calculation process relies on regulations derived from e.g. Basel III accords.

Reasoning with the process (shown in Fig. 2) is provided indirectly over reasoning with regulations encoded as a graph-extended knowledge base. We do not specify regulations in the process itself. The reasoning with the given process is important to check whether the activity of financial service processes is compliant with the Basel regulations upon execution. It means that the activity has been executed successfully and is consistent with regulations from where it is taken [3]. To consume regulations encoded as graph-extended knowledge base, the reasoner must check satisfiability (see Fig. 2).

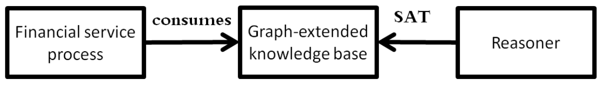


Figure 2. Encoding FSP to graph-extended knowledge base

To encode financial regulations underlying the activities within an FSP space, we use the graph-extended knowledge base, as follows:

(i) Entities related to each activity are encoded as nodes in DG,

(ii) The interconnection between elements is represented by graph roles,

(iii) DG rules and SWRL implement regulations related to an activity,

(iv) Restrictions on the graphs implements control flows between elements.

We encode the regulations into a graph-extended knowledge base for the following reasons:

(i) OWL2 is not expressive enough to encode financial regulations as non-tree structures [17],

(ii) Hyper-tableau algorithm is practically efficient in reasoning with more than one DG. Also it is possible to encode process regulations into more than one DG.

Execution of the *List of Financial Instruments of Core Capital* and the *List of Financial Instruments of Additional Capital* (see Fig. 1) must comply with the following regulations, taken from [3]:

(R1) If a financial institute has a financial instrument as common stock and the common stock can be converted to a currency, then the given currency is a principal amount for the financial institute.

(R2) If a financial institute has a currency as a principal

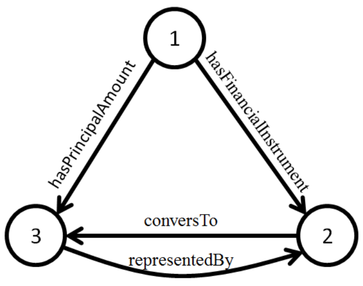


Figure 3. DGs for Common Equity Tier 1 Capital Ratio Calculation Process shown on Fig. 1

amount and the currency is represented by a common stock, then the financial institute has a financial instrument as given by common stock.
The regulations (R1), (R2) are encoded into the following first order logic formulas:

$$(\forall x,y,z)(FinancialInstitute(x) \wedge hasFinancialInstrument(x,y) \wedge convertsTo(y,z) \Rightarrow hasPrincipalAmount(x,z)) \quad (1)$$

$$(\forall x,y,z)(FinancialInstitute(x) \wedge hasPrincipalAmount(x,y) \wedge representedBy(y,z) \Rightarrow hasFinancialInstrument(x,z)) \quad (2)$$

$$(\forall x,y)(FinancialInstitute(x) \wedge convertsTo(x,y) \Rightarrow representedBy(y,x)) \quad (3)$$

Formula (1) encodes regulation (R1) while formula (2) encodes regulation (R2). The formula (3) specifies the relationship between common stock and currency. Formulas (1), (2) can be expressed as property chains that are not allowed in OWL2. Instead, we use formalism based on DGs. Fig. (3) shows the DG which corresponds to regulations (R1) and (R2). The formal definition of the DG shown in Fig. 3 is: $G = (V, E, \lambda)$, where $V = \{1,2,3\}$, $E = \{(1,2),(1,3),(2,3)\}$. Concept names are represented as nodes in the graph (see Fig. 3) as shown in formulation (4) [18]:

$$\lambda\langle 1\rangle = FinancialInstitute; \lambda\langle 2\rangle = CommonStock; \lambda\langle 3\rangle = Currency; \quad (4)$$

The concept *FinancialInstitute* is the main concept. Each node in the graph (see Fig. 3) is related to a concept name. The graph roles are defined as follows:

$$\lambda\langle 1,2\rangle = hasFinancialInstrument; \lambda\langle 1,3\rangle = hasPrincipalAmount; \lambda\langle 2,3\rangle = convertsTo; \quad (5)$$

The extended signature of the graph (see Fig. 3) is:

$$N_C = \{FinancialInstitute, CommonStock, Currency\}, N_{Rt} = \emptyset, N_{Rg} = \{hasFinancialInstrument, convertTo, represnetedBy, hasPrincipalAmount\}, N_I = \{bankOfIrelan, stock1, euro\} \quad (6)$$

N_C represents the set of graph concepts, N_{Rg} is the set of graph roles, and N_I represents the set of individuals. The set of Tbox roles is empty and all roles are graph roles [18]. The formula $G(BankOfIreland, stock1, euro)$ represents an instantiation of the graph.

4.1. Reasoning with Financial Regulations

Reasoning with financial regulations involved with common equity Tier 1 capital ratio calculation process (see Fig. 1) includes:

- Preprocessing the DG which implements regulations consumed by common equity Tier 1 adequacy calculation process,
- Application of derivation rules to the graph-extended knowledge base.

The preprocessing step includes the process of encoding rules for the given DG. It includes rules in program $\mathcal{P}$, equality rules and disjointness rules. In our case, we encode the DG (shown on Fig. 3) into rules according to specification in [18]. Rules in program $\mathcal{P}$ can propagate constraints within DG. The following rules in program $\mathcal{P}$ corresponds to given DG (shown in Fig. 3):

$$hasFinancialInstrument(?x,?y) \wedge conversTo(?y,?z) \rightarrow hasPrincipalAmount(?x,?z) \quad (7)$$
$$hasPrincipalAmount(?x,?y) \wedge representedBy(?y,?z) \rightarrow hasFinacialInstrument(?x,?z)$$

The DG shown on Fig. 3, contains an inverse role i.e. *conversTo* is inverse role of *representedBy* role, so the inverse role can be formalized using the following DG rules:

$$conversTo(?x,?y) \rightarrow representedBy(?y,?x) \quad (8)$$

$$representedBy(?x,?y) \rightarrow conversTo(?y,?x) \quad (9)$$

Formula (10) specifies one equality rule, while the formula (11) specifies one disjointness rule.

$$G(x_1,y_1,z_1) \wedge G(x_1,y_2,z_2) \rightarrow y_1 = y_2 \quad (10)$$

$$G(x_1,y_1,z_1) \wedge G(x_2,x_1,z_2) \rightarrow \perp \quad (11)$$

Finally, the following semantic web rules must be defined:

$$hasFinancialInstrument(?x, ?y) \wedge hasFinancialInstrument(?x, ?z) \rightarrow SameAs(?y, ?z) \quad (12)$$

$$convertsTo(?x, ?y) \wedge convertsTo(?x, ?z) \rightarrow SameAs(?y, ?z) \quad (13)$$

$$hasPrincipalAmount(?x, ?y) \wedge hasPrincipalAmount(?x, ?z) \rightarrow SameAs(?y, ?z) \quad (14)$$

$$representedBy(?x, ?y) \wedge representedBy(?x, ?z) \rightarrow SameAs(?y, ?z) \quad (15)$$

Rules of the form (12), (13), (14), and (15) are important for detecting inconsistency of graph-extend knowledge base shown in Fig. 3. For example, if individual *'BankOfIreland'* occurs in a new DG instance that does not contain individuals *'stock$_1$'* and *'euro'*, then the knowledge base inconsistency will not be detected. We fix this problem by defining all the roles, shown on Fig. 3, functional using rules of the form $R(x, y_1) \wedge R(x, y_2) \rightarrow y_1 = y_2$. After providing these rules, the HermiT reasoner would be able to detect inconsistency.

Applying derivation rules, the hyper-tableau algorithm checks the satisfiability of $(\mathcal{R}, \mathcal{A})$ where $\mathcal{R} = [\sim](\mathcal{G}) \cup \mathcal{P}$, and where $\mathcal{A}$ is defined Abox [17]. To prove satisfiability of given graph-extended knowledge base, the hyper-tableau algorithm tries to construct a model of a $(\mathcal{R}, \mathcal{A})$ by applying different derivation rules to $\mathcal{R}$, $\mathcal{A}$. Before applying the rules, it is important to define at least one Abox $\mathcal{A}$. If $\perp \in \mathcal{A}$ then there is a clash and the algorithm will detect inconsistency in the knowledge base.

5. FraTAct – Activity Transformation Framework

FraTAct (Framework for Transforming Activity) supports the transformation of a nescient activity of a financial service process into an intelligent activity. This framework combines two different but complementary technologies which include *ontology based technologies* and *service oriented technologies*. In this section, an overview of the framework is presented and the underlying technique of the framework is described.

The FraTAct framework is developed relying on Service Oriented Architecture (SOA) [27]. It adopts two characteristics of SOA including *loose coupling* and *ubiquity*. The FraTAct framework consists of a process design interface, a process engine, service component repository, and management components. FraTAct is a multi-layer framework depicted in Fig. 4.

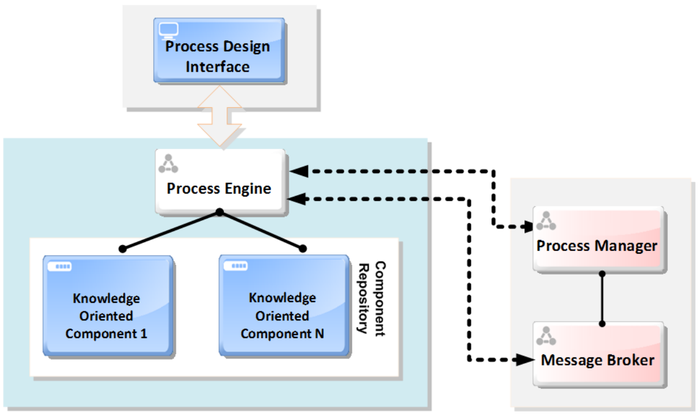


Figure 4. Activity Transaction Framework

In the following, the components of FraTAct framework are briefly described.

(i) *Process Design Interface*: The design interface facilitates the users performing design activities. It offers elements including *activities*, *decision gateways*, etc. for designing financial service business processes.

(ii) *Process Engine*: Process engine is a back-end component which resides at the process layer. A process after designed is deployed in the process engine that executes the process. The process engine executes processes by means of invoking the components for process activities (*aka* services). In other words, the process engine orchestrates the financial services business process activities.

(iii) *Component Repository*: Component repository resides at the component layer. It contains knowledge oriented financial service components that 'actualizes' the process activities. The term 'actualization' refers to *performing operations*. Precisely, the components perform operations of corresponding process activities. The process engine invokes one or more of these financial service components for performing a process activity.

Financial service components are the key constituents in a financial service system. These components perform operations of a corresponding process activity by relying on the financial regulation knowledge base. In effect, a process activity which used to be a simple abstraction that is not aware of financial regulations, turns into an intelligent one. It is worth noting that the intelligence of a financial service process activity is its ability to realize the exact semantics of financial regulations and carry out operations by complying them.

(iv) *Process Manager*: Process manager resides at management layer. It is responsible for performing management tasks such as substituting activities contained in business processes for adapting a change. In addition, it supports modification of business processes.

(v) *Message Broker*: It is a message management component residing at management layer. This component acts to manage messages exchanged between the components, components and process engine, and also between the components. The message broker is connected with the process manager. Any changes occur in processes may promote adjusting the messaging sequences, aggregating messages in message broker.

6. Implementation of FSBA using FraTAct Framework

Throughout this section, we will describe how we have implemented an FSBA using FraTAct framework. According to our understanding, an implementation of a Financial Service based Application(FSBA) will suffice in demonstrating how FraTAct assist in transforming a nescient activity into intelligent activity. Notably, we have implemented common Equity 'Tier 1 Capital Ratio Calculation Process' shown in Fig. 1. Sections include 6.2, 6.3, 6.4 will describe our implementation. In any case, we will provide an overview of the technological aspect of the framework in section 6.1.

6.1. Technological Overview of FraTAct Framework

FraTAct framework shown in Fig. 4 relies on OpenESB [26] that is a middleware comprises a list of components. We will not describe all of these components in this section rather we will briefly introduce the components that are used in FraTAct framework.

FraTAct is built by integrating a process design component. The component provides an interface for designing financial service business process. It offers elements used in designing financial service business processes. The framework integrates a scalable BPEL [21] service engine for executing financial service business processes.

Java Enterprise Edition (Java EE)is integrated into FraTAct. FraTAct Java EE facilitates building ontology oriented financial service components. Since FraTAct is built on the service oriented architecture paradigm, the components are offered as services using Web Service Description Language (WSDL) [34] which describes the service components specifically, the operations, and inbound and outbound messages. FraTAct relies on XML schemas for specifying inputs for the components that perform operations based on the given inputs.

FracTAct was deployed into Glassfish container [9] which is an application server. Furthermore, FraTAct utilizes Composite Application Service Assembly (CASA) to facilitate the users to assemble the financial service components in an FSBA. FraTAct relies on Java Business Integration module (JBI) that offers the binding components used to bind the components, messages, processes, and container. For exchanging messages, FraTAct depends on Simple Object Access Protocol (SOAP) [32].

FraTAct framework utilizes Glassfish administration module as a process manager (shown in Fig. 4) for managing the changes while financial service business processes are running. The module offers a user friendly interface for managing business processes. Moreover, FraTAct integrates the Hermes Java Messaging Service (Hermes JMS) [11] that is a message broker used in publishing and editing messages between service components that hosted at distributed locations.

6.2. Implementation of Service Component

At the first phase of our implementation, we develop service components. The development of service components rely on the mechanism described in section 4. We develop two components `bsOntology.java` and `OntoConsistency.java` that perform the *List Common Equity Tier 1 Financial Instruments* and *Verify the List* activities (see Fig. 1). These activities are regulation intensive, should be performed intelligently. Therefore, the components developed in this paper are ontology based. We describe the development of our ontology based service components in this section.

We implement the following financial instruments: Common shares, Stock surplus, Retained earnings, Comprehensive incomes, Other disclosed reserves, Subsidiaries common shares. Minority interests financial instrument is not part of the current implementation. The Implementation consists of three parts (see Fig. 5):

- Implementation financial regulations as local ontologies. For each financial instrument we have one local ontology. For example, common share local ontology.
- Implementation financial regulations as description graphs.
- Integration ontologies and description graphs.

We use OWL2 language to implement local ontologies. To check consistency of the ontologies we use Pellet reasoner [20]. When OWL2 is not expressive enough for implementing financial regulations, then we

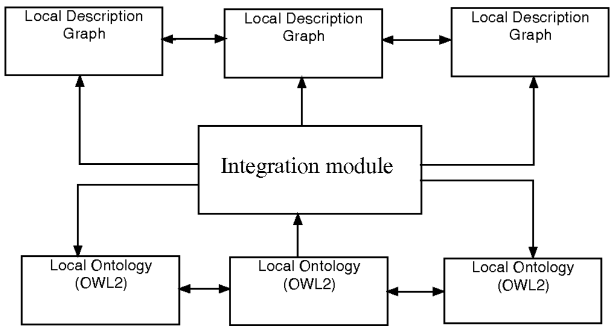


Figure 5. Federation of ontologies in graphs extended knowledge base

use description graphs. To do that, we use the hermit-reasoner API [31] as well as OWL API [25].

To create graph concepts and roles we use *AtomicConcept* and *AtomicRole* classes respectively imported from Hermit API [31]. For the creation of graph roles we allocate memory using the *AtomicRole* class. To check satisfiability of the graph-extended knowledge base, the Hermit reasoner takes ontology and DG as input. In order to do that, we use the *Reasoner* class imported from the Hermit API. For more details, how to implement some financial regulations using description graphs, please see [23]. Imported ontology can not contain built-in predicates because HermiT reasoner does not support reasoning with built-in predicates. For example, the HermiT reasoner is not useful in case of testing consistency of ontology during calculation equity capital in a balance sheet. For such cases, we use Pellet reasoner.

To implement regulations for all financial instruments, integration of ontologies and description graphs is employed. We use a hybrid approach (see Fig. 5) because of the following reasons:

- Local ontologies and graphs are easy to modify. The reason is that some regulations can be changed in the future and it requires modification of local ontologies.

- Integration module connects local ontologies and graphs. It will allow different type of reasoners to check consistency of different ontologies extended with description graphs or just checks the consistency of ontologies without extensions with description graphs.

- Local ontologies and description graphs can be interconnected independently of integration module. For example, one local ontology can be imported into another local ontology.

6.3. Implementation of Financial Service Process

The previous section describes the implementation of service components `bsOntology.java` and `OntoConsistency.java`. We wrapped these components using WSDL [34] as services `bsOntologyService.wsdl` and `bsOntoConsistencyService.wsdl` and published on Glassfish container. In addition, we developed a client side web service `bsOntologyCChkWSDL.wsdl` that provides interface for accepting the requests from the client.

Next, we implemented the *Common Equity Tier 1 Capital Ratio* process shown in Fig 1. In order to implement the process, we used BPEL (version 2.0) - an eXtensible Markup Language (XML) based *de facto* language for developing business processes. First, we defined a *Process Scope* that contains the process activities. The scope contains `StartOntologyProcess` activity that receives requests from client service `bsOntologyCChkWSDL.wsdl`. Fig. 6 shows the connection between `StartOntologyProcess` activity and `bsOntologyCChkWSDL.wsdl` service. The next activity contained in the scope `AssignInput` copies the input to the subsequent activity `ProvideOntology` within the scope. It is a *regulation intensive* activity and upon triggering this activity the process engine invokes the ontology oriented service component `GetOntologyService.WSDL` that actualizes `List Common Equity Tier 1 Financial Instruments` (see Fig. 1). As this component is an ontology based component, it performs activity by realizing the semantics of financial regulations and in effect, the activity `List Common Equity Tier 1 Financial Instruments` turns into an intelligent activity.

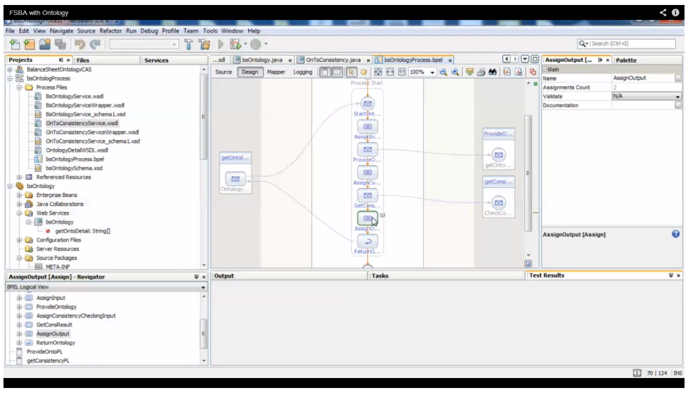
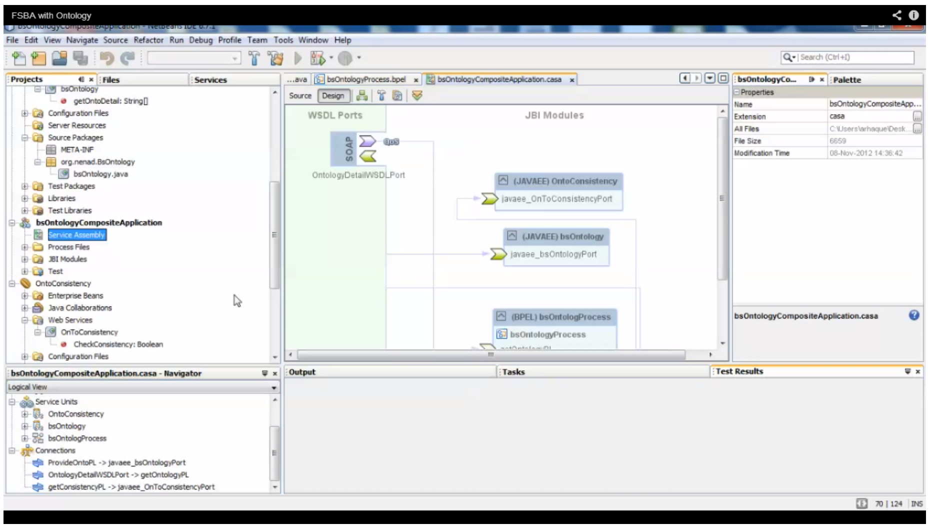


Figure 6. Implementation of Common Equity Tier 1 Capital Ratio Process

Then, we defined `AssignConsistencyInput` activity within the scope that copy the output produced by `ProvideOntology` activity into `GetConsistencyChkResult` which is another regulation intensive activity invokes the `ChkConsistency.WSDL` component actualizes the `Verify the List` activity contained in the



Name	bsOntologyCompositeApp...
Extension	casa
All Files	C:\Users\rhaque\Desk...
File Size	6659
Modification Time	08-Nov-2012 14:36:42

Figure 7. Graphical View of of Financial Service Application

process in Fig. 1. The output is copied by `AssignOutput` to `ReturnOutput` that delivers the output to client service `bsOntologyCChkWSDL.wsdl`. Notably, the start and end events are outside of this scope. Fig. 6 shows the `bsOntologyProcess`.

A BPEL script `bsOntologyProcess.bpel` is generated automatically which is then deployed on the processes engine. The BPEL script is added in Appendix.

6.4. Implementation of Common Equity Tier 1 FSBA

We describe how the financial service based application was developed. The application was developed by binding service components with 'Common Equity Tier 1 Capital Ratio Calculation' process that we implemented(described in the previous section). The 'Composite Application Service Assembly (CASA)' facilitates developing the common equity tier 1 capital ratio calculation service based application. We will describe how we assembled the service components into an application.

We generated the Jar files from process `bsOntologyProcess.bpel` and service components `bsOntology.java` and `OntoConsistency.java`. The jar files `bsOntologyProcess.jar`, `bsOntology.jar`, and `OntoConsistency.jar` are then imported on Java Business Integration module that binds the service components using SOAP protocol. The application was then built and deployed in the Glassfish application server. Fig. 7 depicts the development of common equity tier 1 FSBA.

The financial service based application of common equity calculation process is now ready to automate the common equity tier 1 capital ratio calculation process. The application performs the regulation intensive process activities intelligently.

Scalability and performance of the approach on a large scale of DGs depends on the HermiT reasoner performances discussed in [19]. The reasoner shows significant advantages in reasoning with a large scale ontologies [19] comparing to other standard reasoners such as the Pellet [20]. The open issue is how to query financial ontologies extended with description graphs by using SPARQL.

7. Related Works

In this section, we discuss the related solutions revolving around the financial regulation knowledge base, reasoning, and process. To the best of our knowledge, financial regulation has been scoped in a very small amount of research. Thus, the actual need (e.g., machine readable definition of Basel-based regulations) of the financial service industry has been addressed very briefly in current solutions. Some very recent research including [29], [30], and [1] focus on investigating the issues and challenges, and defining a roadmap for governance, risk and compliance within business processes. Nonetheless, no concrete solution has been discussed in these works. Additionally, a language to capture the compliance requirements of business processes is proposed in [33] and [6]. The language is promising, but offers a very limited number of constructs that do not facilitate defining financial regulations in FSB. Solutions including [7] and [16] are offered to perform reasoning on business processes to verify the correctness of business processes. However, these solutions do not address the problem of this research.

8. Conclusion

In this paper, we have shown how to transform a nescient activity within an FSP into an intelligent activity. In order to support the transformation, we have shown how to encode financial regulations into a graph-extended knowledge base that assists process activities to perform operations intelligently. Additionally, we have shown how to perform reasoning with financial regulations. We have also described the current state of implementation.

We plan to extend the solution in order to enable a system to decide actions for the conditions that are evolved at runtime. In the future, we plan to cover more complex financial regulations. In addition, our intention is to cover financial regulations from various bodies. We also plan to enable these systems to support fuzzy reasoning. Keeping completeness for an incomplete ontology reasoner is a challenge [8] which we have experienced in this research. We will try to tackle this challenge in its future extension.

Currently, FraTAct is a specific framework which serves only financial domain but we plan to add more features that can cover other domains such as business to business(B2B). In B2B, businesses are governed by service-level agreement (SLA). The business processes activities must be accomplished by satisfying SLAs. We plan to develop knowledge-oriented component which provides explicit semantics of SLA clauses for corresponding process activities. We are considering to integrate technologies such as Semantic Markup for Web Services(OWL-S) [24].

Acknowledgement. This research is done within the scope of the Governance, Risk, and Compliance Technology Centre project funded by Enterprise Ireland. The authors thank to Patrick O'Sullivan for his contribution to improve the readability of the paper. The authors thank to the GRCTC Team for supplying inputs related to financial instruments and the Basel Accords. The authors is thankful to Boris Motik from University of Oxford, UK, for his advices in practical applications of DGs in modeling financial regulations.

References

[1] Abdullah S.A., Sadiq W.S., and Marta I. (2010) Emerging Challenges in Information Systems Research for Regulatory Compliance Management. *International Conference on Advance Information System Engineering(CAiSE)*. pp. 251-265.

[2] Apache Jena, http://jena.apache.org/documentation/ontology/index.html

[3] Basel III (2010) A Global Regulatory Framework. *Bank for International Settlement*.

[4] Cuenca Grau B., Horrocks I., Motik U., Parsia P., Patel-Schneider P.F., and Sattler U. (2008) OWL 2: The next step for OWL. *Journal of Web Semantics*. Vol 6(4). pp. 309-322.

[5] IFRSs and XBRL, http://www.ifrs.org/XBRL/XBRL.htm,Lastvisited:May, 2012.

[6] Elgammal E., Turetken O., and Heuvel W.J.V.D. (2010) Using Patterns for the Analysis and Resolution of Compliance Violations. emphInternational Journal of Cooperative Information System. vol. 21(1), pp. 31-54.

[7] Francescomarino C. D., Ghidini C., Rospocher M., Serafini L., and Tonella P. (2008) Reasoning on Semantically Annotated Processes. *International Conference on Service Oriented Computing*. pp.132-146.

[8] Grau B.C., Motik B., Stoilos G., Horrocks I. (2012) Completeness Guarantees for Incomplete Ontology Reasoners: Theory and Practice. *Journal of Artificial Intelligence Research*. vol. 43. pp. 419-476.

[9] Java Community Process (2012). Glassfish Container. Available on: http://glassfish.java.net/

[10] Haque R., Krdzavac B.N., Butler T., (2012) Transforming Nescient Activity into Intelligent Activity. *In Proc. of the Data and Knowledge Engineering (ICDKE 2012)*. isbn:978-3-642-34678-1.

[11] Hermes. Hermes Java Messaging Services(Hermes JMS). Available on: http://www.hermesjms.com/confluence/display/HJMS/Home.

[12] Healy M.P., and Palepu G.K. (2003) The Fall of Enron. *Journal of Economic Perspectives*. vol. 17(2). pp. 3-26. Spring Publication.

[13] Horrocks I., Kutz O. , and Sattler U.(2006) The Even More Irresistible SROIQ. *In Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*. pp. 57-67. (AAAI Press).

[14] Horrocks I., Patel-Schneider P.F., Boley H., Tabet S., Grosof B., and Dean M.SWRL: A Semantic Web Rule Language Combining OWL and RuleML. *W3C Member Submission*.

[15] Leymann F. and Roller D. (1999) *Production Workflow: Concepts and Techniques*. Prentice Hall; Edition 1, ISBN-10: 0130217530.

[16] Missikoff M., Proietti M.,and Smith F. (2010) Reasoning on Business Processes and Ontologies in a Logic Programming Environment. *In the Proceedings of 3rd Interop-Vlab.It Workshop, CEUR-WS*. pp. 653.

[17] Motik B., Shearer R., and Horrocks I.(2009) Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*. pp.165–228.

[18] Motik B., Cuenca Grau B., Horrocks I., and Sattler U. (2008) Representing Structured Objects using Description Graphs. In Gerhard B. and *Jer^{0}me* L., editors. *In Proc. of the 11th Int. Joint Conf. on Principles of Knowledge Representation and Reasoning (KR)*. pp. 296–306. AAAI Press, Sydney, NSW, Australia.

[19] Shearer R., Motik B.,and Horrocks I.. (2008) HermiT: A Highly-Efficient OWL Reasoner. In Ruttenberg A., Sattler U., and Dolbear C., editors. *In Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*. Karlsruhe, Germany.

[20] Pellet reasoner. http://clarkparsia.com/pellet/

[21] OASIS (2007). Business Process Execution Language. Available on: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[22] OMG (2011). Business Process Modeling Notation. Available on:http://www.omg.org/spec/BPMN/2.0/

[23] N. Krdzavac, R. Haque, T. Butler, Web-based Reasoning With Balance Sheets, Workshop 'AI on the Web' at the 35th German Conference on Artificial Intelligence (KI2012, AIW 2012), 09/2012, Saarbrucken, Germany, (2012)

[24] OWL-S. Available: http://www.w3.org/Submission/OWL-S/. Last visited May, 2012.
[25] OWL API. Available: http://owlapi.sourceforge.net/. Last visited May, 2012.
[26] OpenESB Community (2012). Open Enterprise Service Bus (OpenESB). Available on:http://www.open-esb.net/
[27] Papazoglou P.M., Heuvel v.d.W., (2007) Service oriented architectures: approaches, technologies and research issues. *VLDB Journal*. vol. 16(3). pp. 389-415.
[28] Polyvyanyy A., and Weske M. (2008) Hypergraph-Based Modeling of Ad-Hoc Business Processes. *Business Process Management Workshops*. pp. 278-289.
[29] Sadiq W.S., Muehlen Z.M., and Indulska M. (2012) Governance, risk and compliance: Applications in information systems. *Information Systems Frontiers*. vol.14(2). pp. 123-124.
[30] Sadiq W.S. (2011) A Roadmap for Research in Business Process Compliance. *In proceedings of BIS (Workshops)*. pp. 1-4
[31] Shearer R., Motik B., and Horrocks I.(2008) HermiT: A Highly-Efficient OWL Reasoner. *In Alan Ruttenberg, Ulrile Sattler, and Cathy Dolbear, editors, Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*. pp. 26 - 27. http://hermit-reasoner.com/
[32] SOAP (2007). Simple Object Access Protocol(SOAP). Available on:http://www.w3.org/TR/soap/
[33] Turetken O., Elgammal A., Heuvel W.J.V.D., and Papazoglou P.M. Capturing Compliance Requirements: A Pattern-Based Approach. *IEEE Software*. vol.29(3). pp. 28- 36.
[34] WSDL (2001) Web Service Description Language. Available on: http://www.w3.org/TR/wsdl

9. Appendices

The BPEL script `bsOntologyProcess.bpel` is presented in the following.
BPEL Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<process
    name="bsOntologyCChkProcess"
    targetNamespace="http://enterprise.netbeans.org/bpel/
    bsOntologyConsChkProcess/bsOntologyCChkProcess"
    xmlns:tns="http://enterprise.netbeans.org/bpel/
    bsOntologyConsChkProcess/bsOntologyCChkProcess"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/
    executable"
    xmlns:sxt="http://www.sun.com/wsbpel/2.0/process/
    executable/SUNExtension/Trace"
    xmlns:sxed="http://www.sun.com/wsbpel/2.0/process/
    executable/SUNExtension/Editor"
    xmlns:sxeh="http://www.sun.com/wsbpel/2.0/process/
    executable/SUNExtension/ErrorHandling" xmlns:sxed2=
    "http://www.sun.com/wsbpel/2.0/process/executable/
    SUNExtension/Editor2"
    xmlns:ns0="http://xml.netbeans.org/schema/
    bsOntChkInputSchema">
<import namespace="http://j2ee.netbeans.org/wsdl/
bsOntologyConsChkProcess/bsOntologyCChkWSDL"
location="bsOntologyCChkWSDL.wsdl" importType=
"http://schemas.xmlsoap.org/wsdl/"/>
<import namespace="http://enterprise.netbeans.org/
bpel/OnToConsistencyServiceWrapper"
location="OnToConsistencyServiceWrapper.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
<import namespace="http://OnToConsistency.nenad.com/"
location="OnToConsistencyService.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
<partnerLinks>
   <partnerLink name="ProvideOntChkResultPL"
   xmlns:tns="http://enterprise.netbeans.org/bpel/
   OnToConsistencyServiceWrapper" partnerLinkType=
   "tns:OnToConsistencyLinkType" partnerRole=
   "OnToConsistencyRole"/>
   <partnerLink name="OntologyConsChkPL"
   xmlns:tns="http://j2ee.netbeans.org/wsdl/
   bsOntologyConsChkProcess/bsOntologyCChkWSDL"
   partnerLinkType="tns:bsOntologyCChkWSDL"
   myRole="bsOntologyCChkWSDLPortTypeRole"/>
</partnerLinks>
<variables>
   <variable name="CheckConsistencyOut" xmlns:tns=
   "http://OnToConsistency.nenad.com/" messageType=
   "tns:CheckConsistencyResponse"/>
   <variable name="CheckConsistencyIn"
   xmlns:tns="http://OnToConsistency.nenad.com/"
   messageType="tns:CheckConsistency"/>
   <variable name="BsOntologyCChkWSDLOperationOut"
   xmlns:tns="http://j2ee.netbeans.org/wsdl/
   bsOntologyConsChkProcess/
   bsOntologyCChkWSDL" messageType=
   "tns:bsOntologyCChkWSDLOperationResponse"/>
   <variable name="BsOntologyCChkWSDLOperationIn"
   xmlns:tns="http://j2ee.netbeans.org/wsdl/
   bsOntologyConsChkProcess/bsOntologyCChkWSDL"
   messageType="tns:bsOntologyCChkWSDLOperationRequest"/>
</variables>
<sequence>
   <receive name="StartOntoChkProcess"
   createInstance="yes"
   partnerLink="OntologyConsChkPL"
   operation="bsOntologyCChkWSDLOperation"
   xmlns:tns="http://j2ee.netbeans.org/wsdl/
   bsOntologyConsChkProcess/
   bsOntologyCChkWSDL"
   portType="tns:bsOntologyCChkWSDLPortType"
   variable="BsOntologyCChkWSDLOperationIn"/>
   <assign name="AssignInput">
      <copy>
         <from>$BsOntologyCChkWSDLOperationIn.InputPart/
         ns0:getConsChkResult</from>
         <to>$CheckConsistencyIn.parameters/
         getConsChkResult</to>
      </copy>
   </assign>
   <invoke name="ProcessCChk" partnerLink=
   "ProvideOntChkResultPL"
   operation="CheckConsistency" xmlns:tns=
"http://OnToConsistency.nenad.com/" portType=
"tns:OnToConsistency"
inputVariable="CheckConsistencyIn"
outputVariable="CheckConsistencyOut"/>
<assign name="AssignOutput">
   <copy>
      <from>$CheckConsistencyOut.parameters/
      return</from>
      <to>$BsOntologyCChkWSDLOperationOut.OutputPart
      /ns0:ConsChkResult</to>
   </copy>
</assign>
```

```
      <reply name="ReturnOntChkOutcome"
      partnerLink="OntologyConsChkPL"
      operation="bsOntologyCChkWSDLOperation"
      xmlns:tns="http://j2ee.netbeans.org/wsdl/
      bsOntologyConsChkProcess/
      bsOntologyCChkWSDL"
      portType="tns:bsOntologyCChkWSDLPortType"
      variable="BsOntologyCChkWSDLOperationOut"/>
   </sequence>
</process>
```