International Journal of Scientific & Engineering Research, Volume 4, Issue ƛ,ɯ)Ü0à 2013
ISSN 2229-5518

Transformation of Business Processes into UML Models: An SBVR Approach

Amit Raj, Ashish Agrawal, T. V. Prabhakar

Abstract— A business process is a set of activities that collectively perform and deliver a complex functionality. At a very early stage of software development, business semantics are discussed with stakeholders. A business analyst writes these semantics in his preferred language such as English. A software architect/manager understands them and creates business process. An IT team converts these business processes into platform-specific models such as UML, which are ultimately converted into the application code. During this life cycle, business semantics are passed through several stages that dilute the semantics. This paper presents a SBVR approach to design business process in structured English and illustrates a mechanism to automatically convert those business processes into UML models that can be further used for application-specific code generation.

Index Terms— Business rules and modeling, MDA, Model Transformation, Metamodeling, SBVR, UML.

———————————— ◆ ————————————

1 INTRODUCTION

THE Business Process Modeling and Business Rule Modeling are the most prevalent approaches for modeling behavior of business. In process modeling, operational behavior of an enterprise is represented through process models. Rule modeling approach follows the methodology of separating business rules from process models and defining rule models as static nature of business. Following business rules approach [1], Object Management Group (OMG) has given Semantics of Business Vocabulary and Business Rules (SBVR) [2], a declarative meta-model for describing business vocabulary and business rules. Benefits of SBVR are its declarative nature, rule modeling approach, natural language representation and formal logical backbone. However, process related concepts are outside the scope of SBVR.

Both the process modeling and rule modeling approaches have different benefits in terms of semantic representation. Representational analysis done by Michel et al. [3], show that any single approach is not capable of representing all business constructs. Rule modeling approach focuses on decision points which regulate the business. On the other hand, process modeling approach tries to minimize the amount of work required in processing, but ignores decision points. Michel et al. showed that combination of a rule modeling language and a process modeling language covers maximum representational constructs.
From control flow perspective, process modeling can be categorized into two types:

1. **Process Modeling** defines a process as an exact sequence of process elements (tasks, events, etc.) and routing elements (gateways). Examples of procedural process modeling languages are; BPMN [4], BPEL [5], EPC [6], WorkFlowNets [7], UML Activity Diagram [8], etc.
2. **Declarative Modeling** defines a process as a set of process elements and declarative statements representing constraints over them. Examples of declarative process modeling languages are; Case Handling [9], Penelope [10], ConDec [11], DecSerFlow [12], Constraint Specification Framework [13], etc.

In procedural modeling, structure and execution path of a process is fixed at design time. Compared to this, a declarative process model only defines "what" this process offers to the business, leaving out the information of "how" to achieve it. In practice, a designed process model might be executed in different operating conditions of business. In knowledge intensive business processes, sometimes it is not feasible to specify exact execution path of a process at design time. Process models also evolve as a result of process analysis and process improvement. Thus, for knowledge intensive and dynamic processes, flexibility and adaptability are essential requirements of process models.

The most popular and widely used notation for process modeling is BPMN due to its simple graphical notation, which is closer to business people. However, process models prepared using BPMN are procedural in nature and contain the same problems mentioned above. There also exist several declarative languages in literature but most of them cover only a specific part of the process model. For example, Penelope only describes sequence and timing constraints of a process model. ConDec mainly focuses on control flow constraints. Case-Handling mainly targets data-constraints and authorization. To overcome these problems, there is a need for a generic declarative process meta-model which can be integrated with rule meta-model (SBVR) and its transformation into platform specific models such as UML models. This work is an extension of

————————————————

- *Amit Raj is currently pursuing PhD degree program in School of Computer Science and Statistics at Trinity College of Dublin, Ireland E-mail: araj@scss.tcd.ie*
- *Ashish Agrawal is currently pursuingPhD degree program in Department of Computer Science and Engineering at Indian Institute of Technology, Kanpur, India, E-mail: agrawala@cse.iitk.ac.in*
- *T.V. Prabhakar is a professor in the department of Computer Science and Engineering at Indian Institute of Technology, Kanpur, India Email: tvp@cse.iitk.ac.in*

our previous works [14] [15], where we provide a mechanism to define the business processes in SBVR and their transformation into UML models. We have applied the SBVR approach to model business processes. A generic meta-model for declarative business process modeling, Semantics of Business Process Vocabulary and Process Rules (SBPVR) is proposed. SBPVR provides conceptual vocabularies to define process elements, their semantics and process rules. SBPVR also supports natural language representation (Structured English) for the process models.

On the other hand, there is a need to automatically convert such business semantics into the UML models as SBPVR models will be required to generate an application code. Another contribution of this paper is to bridge the gap between business people and IT people in order to minimize the loss of semantics. The paper shows how to create the business processes in SBPVR and also shows how to transform them to PIMs. Mark H. Linehan [16] had tried to develop PIMs from SBVR models. Markus Schacher [17] shows the transformation of business rules to executable UML Models (also called xUML Models) based on their CASSANDRA Platform [18] and also shows how BPMN models can be mapped to xUML Models. Eriksson et. al., in his book [19] has shown the business modeling with UML.

Main challenge in transforming the Business processes rules to PIM is the detection of automatable business rules and their automation. "Automating" a rule means to enforce the rule through automation. In general, an enforcement policy needs to be specified for each rule and putting an obligation on the system or process as to exactly how, when, and where the system or process will enforce the rule. That is, there should be a rule or a set of rules for each automatable rule about how each automatable rule will be enforced by the system and in this paper we are trying to describe that set of rules. This is often non-trivial, as there are often several options available, and it is a system design choice which one to use. Additionally, the enforcement may be complex, involving many steps and coordinated activity to enforce the rule. This information is generally not in the automatable rule itself, but involves other considerations too. The contribution of this paper is to analyze all those requirements and presenting a methodology which allows business people to convert their business processes into UML Activity Diagram (AD), Sequence Diagram (SD) and Class Diagram (CD).

2 Related Work

2.1 SBVR

SBVR categorizes business knowledge into three parts; Concept types, Fact types and Business rules. Methodology to create SBVR models follows Business Rules Mantra, "Rules are built over fact types and fact types are based on concepts" [2] [1]. Figure 1 depicts the methodology of SBVR. It also supports natural language representation Structured English which enhances its usability for business people. A graphical representation has also been developed by Musham et al. [20].

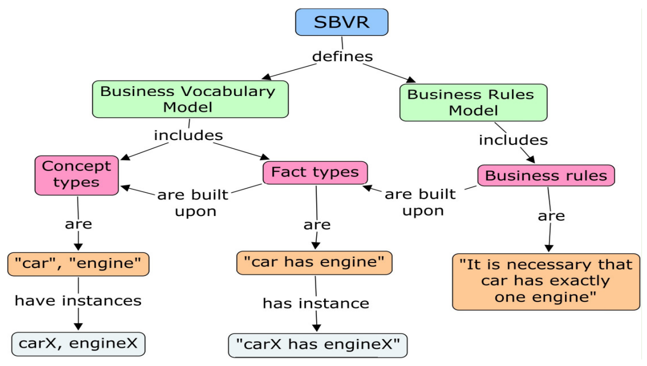

Figure 1. SBVR Methodology

2.2 Existing BP Modeling Languages

There exists several process modeling languages in the literature. These languages differ in the terms of procedural or declarative nature, target area, audience, representation, formalization etc.

BPMN has become de-facto standard for most of the BPM suites due to its simple graphical notation. Languages like BPEL and UML Activity Diagram, which target system model, are too complex to be used by business people. Models defined using BPMN are procedural in nature but lack in semantics. Languages like WorkFlowNets [7] and EPC [6] are supported by mathematical formalization (petri-nets, pi-calculus etc.). The downsides are their procedural nature and limited coverage of various aspects of process modeling.

Compared to procedural languages, most of the declarative languages cover only a specific part of a process model. Sadiq et al. [13] proposed a constraint specification framework for flexible business processes. This approach can be viewed as a starting point for declarative modeling as only a part of the process model is described declaratively. Van der Aalst et al. [9] have given a declarative paradigm to model flexible business processes where execution of the process depends upon the current case i.e. data produced by the process so far. However, it is very restricted due to data-driven approach and does not cover all possible real scenarios. Pesic et al. [11] have given a declarative language ConDec, based on Linear Temporal Logic which consists of tasks and constraints (mainly for control flow). Graphical representation of ConDec is also complex to be used by business people. Goedertier et al. [10] have presented a declarative language, Penelope which mainly focuses on sequence and timing constraints of a process model.

2.3 Business Process to UML

Mark H. Linehan [16] explores the work to specify the semantics and rules in SBVR as extension of business models that are automatically translated to PIMs which in turn get converted to PSMs. This technology is known as Model Driven Business Transformation (MDBT). Here, PIM models include UML Class diagram, State Chart and Use Case Diagram. But the paper doesn't explicitly specify the algorithms and there is no such information of how to find the function of the classes too.

International Journal of Scientific & Engineering Research Volume 4, [illegible]
ISSN 2229-5518

Markus Schacher [17] explores the work to view the Business rules in the perspective of SBVR and CASSANDRA [18]. This paper develops an environment completely based on the CASSANDRA platform to create executable UML Models (also called xUML Models) from the Business Rules. It also provides a rule-set to transform SBVR Vocabulary and Rules into Class Diagram, Use Case Diagram based on xUML platform. This paper also shows how the business activities represented in BPMN can be transformed to xUML Notation. Dane Sorensen et al., in his study [21] shows the lack of ontology in the SBVR Metamodel and also shows how ontology integration into SBVR could improve the future releases of this standard. SBeaVer [22] is an open source SBR tool created by Maurizio De Tommasi and Pierpaolo Cira at the University of Lecce'in Italy, in a project funded by the European Digital Business Ecosystem [23] project. This tool runs as a plug-in for Eclipse platform which enable the user to create, validate and verify the business vocabulary and rules but this tool neither generates the logical formulation nor any platform independent model.

3 SBVR Background

SBVR is an abbreviated form of "Semantics of Business Vocabulary and Rules" which has been accepted by OMG in 2005. OMG's Model Driven Architecture (MDA) [24] is a multi layered approach to implement a real world application. SBVR is completely compatible with MDA and behaves as a Computational Independent Model. Compatibility with MDA increases its adoption by several business organizations. SBVR is an approach to allow the business analysts or any business person, who is interested in writing the business rules, to express the business artifacts in natural language format. A business person can write them in his own language and can create a semantic model for it. This semantics model could be same for different business designs in different languages because semantic metamodel of SBVR is totally independent of representation [2].

The basic mantra of SBVR is "Rules are built of fact types and fact types are built of terms" which is clearly described in Figure 2 with example.

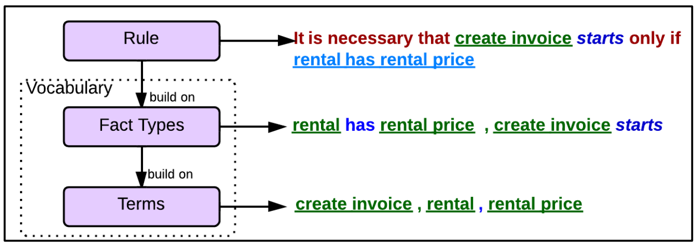
Rule
It is necessary that create invoice starts only if rental has rental price
Vocabulary
build on
Fact Types
rental has rental price , create invoice starts
build on
Terms
create invoice , rental , rental price

Figure 2. SBVR Schema Model

SBVR has its own set of keywords and terminology to write the business vocabulary and rules. Following is a little introduction with its terminology given in [2].

3.1 SBVR Business Vocabulary

SBVR Business Vocabulary is the collection of business entities, their instances and relationships between them, which can be used by any organization in their writing and talking during the course of their business.

1. **Terms:** These are the noun or group of words which can be collectively used for the designation of a business entity. For example: "car rental agency".
2. **Name:** These are the words which are used to represent the instance of a particular term. Eg. Hertz that is an instance of car rental agency.
3. **Fact Type:** These are the sentences which represent the relationship between terms. We are using the template term-verb-term to establish the relation between two terms, as it is very obvious that a mutual relationship between three entities can be easily break down to maximum of three binary relations. For example, the fact type "customer owns account is member" states that a customer is related with account and account is related with member and a person who owns an account will be a member. This relationship can be breakdown to two relations as described by the two fact type like "customer owns account" and "customer is member".

3.2. SBVR Business Rules

These are the sentences under business jurisdiction which guide the structure and behavior of an organization.

1. Structural Rules: These are the rules which represents how the business should be organized
2. Operative Rules: These are the rules which govern the business execution.

3.3. Parsing SBVR Business Vocabulary & Rules

SBVR allows the following four types of expressions as given in [2], to create the business vocabulary and rules. To parse the business model given in SBVR, we have to just identify the following four types of expressions:

1. **Term:** A noun or a group of words which collectively designate a basic unit of knowledge. They always start with small letters.
2. **Name:** These are the nouns or group of words which starts with capital letter.
3. **Verb:** For parsing the verbs, there are two methods, either create a list of all the existing verbs or ignore the terms in fact types (term-verb-term) and get the verb.
4. **Keyword:** There are some inbuilt keywords given in [2]. Create a list of all the keywords and point out them in the sbvr sentences.

3.4. Logical Formulation

Every rule presents some semantics of a business artifact. SBVR provides a structure to formulate that semantic which is known as logical formulation. It is an abstract and language independent syntax to represent the meaning of a rule as described in Figure 3. Some of the terminology used in Figure 3 is not described in the paper but properly explained in [2].

SBVR Specification [2] also provides a Meta Object Facility (MOF) [25] representation of Logical Formulations.

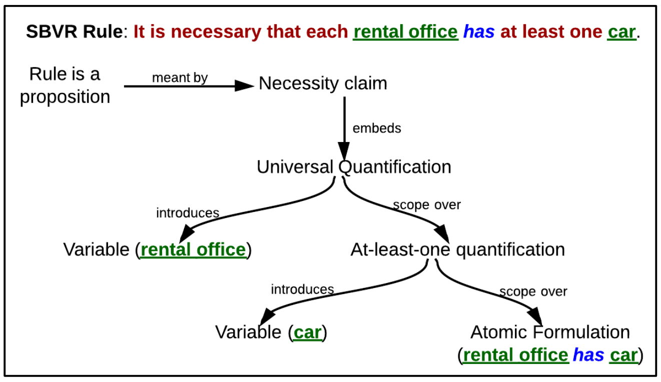

Figure 3. Logical Formulation of a SBVR Rule

4 SBPVR: Semantics of Business Process Vocabulary & Process Rules

SBPVR follows the methodologies of SBVR [2] and Business Rules Approach (BRA) [1] to describe process models. Following the fact-oriented approach, SBPVR divides process knowledge into three parts; Process concept type, Process fact type and Process rules. Figure 4 explains these categories with instances from real world.

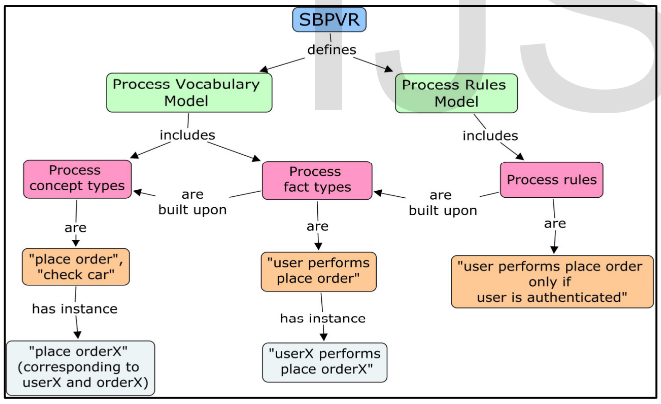
SBPVR
defines
Process Vocabulary Model
Process Rules Model
includes
includes
Process concept types
are built upon
Process fact types
are built upon
Process rules
are
are
are
"place order", "check car"
"user performs place order"
"user performs place order only if user is authenticated"
has instance
has instance
"place orderX" (corresponding to userX and orderX)
"userX performs place orderX"

Figure 4. SBPVR Methodology

Definitions of these categories are:

1. **Process Concept Type:** represents a dynamic entity in the process model. For example, task, event, interaction etc. Instance of a process concept type represents the state of affairs happening in the business.
2. **Process Fact Type:** represents either characteristic of a process concept type (unary fact type) or relation between two or more process concept types (binary or n-ary fact type). Instance of a fact type represents relation between instances of process concept types.
3. **Process Rule:** define constraints over the structure and flow of the process.

4.1 Vocabulary for Describing Business Process Vocabulary

This section specifies the vocabulary to be used to describe elements in the process model. Elements of process model and their semantics are derived from existing work in the field of process modeling (BPMN[4], Penelope [10], EPC [6], BPEL [5], Case Handling Paradigm [9]) and in the field of process specification (PSL [26] and ebXML [27]). To provide a generic and extendable meta-model, these elements and their semantics are categorized in a hierarchical fashion. This categorization is described in the next subsections.

4.1.1 Process Concept Type

Process concept type is the abstract class for all dynamic entities in the process model. Semantics of a process concept are its relations to the happening on its enactment and changes occurred due to its performance. Happening is the meaning of the activity of change in business. For example, meaning of a task type, "Place Order" is the actuality of an instance of Customer giving Order to an instance of Salesman. A process concept can be defined in the context of a business process. A process concept type can generalize or specialize another process concept type. Each process concept incorporates characteristics which make it unique in the model. Examples of these characteristic are, "Concept Type" (task, event), "State" (Start, Error, Finish) etc. Semantics of a process concept are independent of the way of its execution. Figure 5 shows the categorization of process concept type as explained in next subsections.

4.1.2 Work Type

Work type specializes process concept type and formulates work to be performed or coordinated by an agent or a coordinator respectively. Work type is a general concept for the elements which involve work to be performed (tasks, activities and whole processes). In addition to characteristics of process concept type, work type has a role binding with an agent (SBVR: concept type). A work type may belong to one speech community or can represent collaborative work between two speech communities. It has a place holder for its owner (SBVR: concept type). A work type has a purpose associated with it. Work type has a placeholder for business rules which guide its enactment. Output of work type can be represented in terms of changes in state model. Following are the specializations of work type on the basis of granularity:

1. **Task Type:** represents unit amount of work in the process model.
2. **Activity Type:** represents a set of tasks, interactions and events
3. **Individual Process:** represents a complete process which achieves some business goal or provides some service.

4.1.3 Interaction Type

Interaction type specializes task type in which a business document (message or artifact) is being transferred during its enactment. For the generalization of both orchestration and

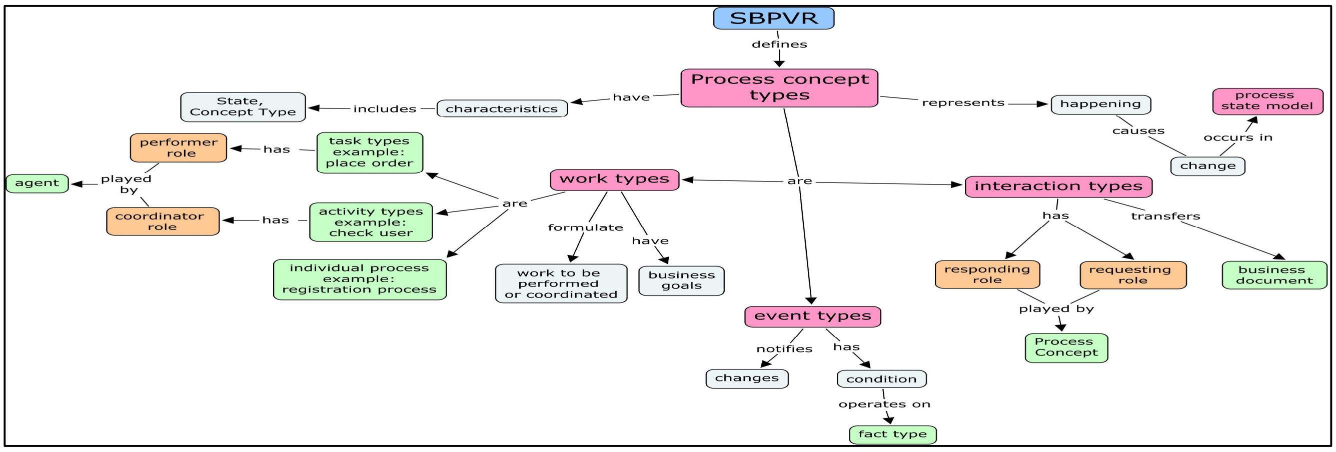

Fig. 5. SBPVR Process Concept Types

choreography processes, interactions are first class concepts in the SBPVR. An interaction has two role bindings, requesting role and responding role associated with it. This binding represents transfer of business document from responding role to requesting role.

4.1.4 Event Type

Event notifies changes in the state model to process concepts. Happening of an event is instantaneous. Event has triggers associated with it which cause its occurrence in the process.

4.1.5 Event Type

State model represents state of the business at one particular point of time. SBPVR models are represented by process schema, which includes process concept types, process fact types and *SBVR:conceptual schema*. State model is an instance of process schema in one possible world scenario at one point of time.

4.2 Process Fact Type

A process fact type is a relation between two concept types (at least one is process concept type.jpg). The relation can be between a dynamic entity (process concept type) and a static entity (SBVR:Concept type) or only between dynamic entities. A process fact type is based on a verb concept (process fact type role) which binds two or more entities in semantic relation. Semantics associated with a process fact is the kind of relationship between concepts. SBPVR divides these relationships in seven categories. These categories and respective examples are shown in Figure 6.

4.3 Process Rules In SBPVR

In SBPVR, process rules are statements which constrain or guide business behavior in the context of business processes. In traditional procedural languages, these constraints or design decisions are either hidden or defined implicitly. In SBPVR, process rules are extracted from the process diagram and represented separately. A process rule can have two kinds of guidance (based on SBVR's categorization of guidance):

1. **Structural Guidance:** claims necessity on the structure of process models and cannot be violated during the process enactment.
2. **Operative Guidance:** claims obligation on the behavior of process models.

Another popular categorization of rules is given by Taveter el al. [28]. They have classified rules into four categories; Integrity constraints, Derivation rules, Reaction rules and Deontic assignment. Muninder et al. [29] have shown that business protocols can be described using commitments over the agents. Hay [30] has categorized rules into structural assertion, action rules and derivation rules. He also included authorization rules. Goedertier et al. [31] have given sixteen types of rules divided into three categories (Control flow, Data Aspects and Organizational Rules). These categories are influenced by work of Jablonski et al. [32]. SBPVR does not aim to restrict itself to few specific types of rules but to provide an abstract classification of process-aware business rules.

In SBPVR, process rules are categorized into five categories, influenced from the work of Wagner et al. [28]. Since SBPVR only targets the rules which are defined in the context of a process, definitions and boundaries of above categories are defined in the context of a process. Following subsections describe these five categories and for each category, they also specify few kind of types belonging to that category. SBPVR does not aim to restrict itself to only the mentioned types and they can be extended by enterprises. Figure 7 shows the categorization of process rules in SBPVR.

4.3.1 Integrity Rule

Integrity rule constrains flow or integrity of a process model. In SBPVR, dynamic entities and state transitions are represented by concept types and fact types respectively. Integrity rules guides to maintain integrity of state model in the operations where instances are added or deleted from state model. Following are few types of integrity rules:

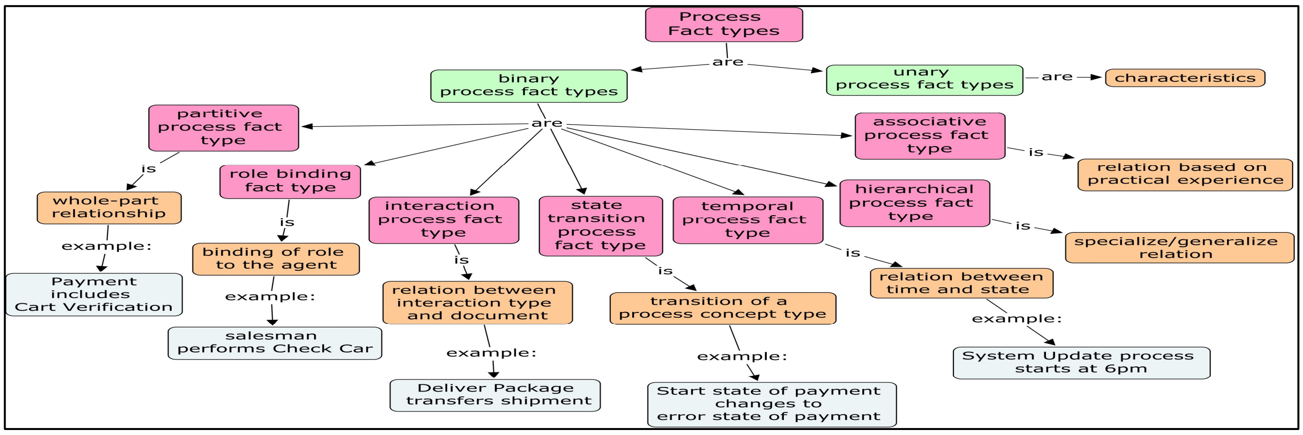
Process Fact types
are
binary process fact types
unary process fact types
are
characteristics
partitive process fact type
are
associative process fact type
is
relation based on practical experience
is
whole-part relationship
role binding fact type
interaction process fact type
state transition process fact type
temporal process fact type
hierarchical process fact type
is
specialize/generalize relation
example:
is
binding of role to the agent
is
is
is
relation between time and state
Payment includes Cart Verification
example:
relation between interaction type and document
transition of a process concept type
example:
salesman performs Check Car
System Update process starts at 6pm
example:
example:
Deliver Package transfers shipment
Start state of payment changes to error state of payment

Figure 6. Process Fact Types in SBPVR

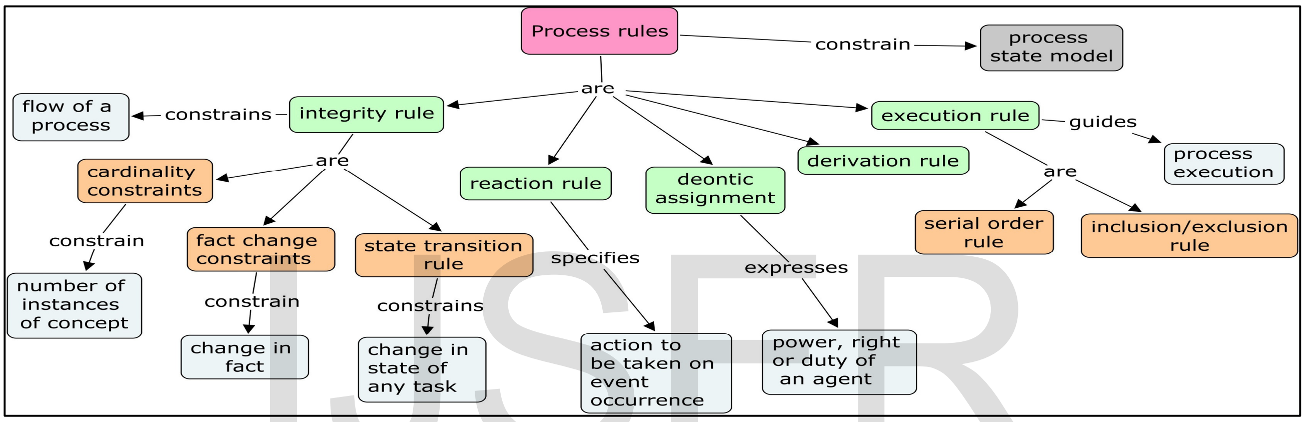

Fig. 7. Process Rules in SBPVR

1. **State Transition Constraint:** This rule controls transition of a process concept from one state to another. Constraints are formulated by existence or non-existence of facts in the state model.
   Example: *It is necessary that accept order starts only if user has order and user is valid (activity precondition).It is necessary that accept order is completed only if order is accepted(activity post condition).*
2. **Fact Change Constraint:** This type of rule constrains the change of fact types in state model.
   Example: *It is not possible that fact type, order is rejected, changes to order is accepted.*
3. **Cardinality Constraint:** This type of rule constrains the number of instances of a concept type in the context of another concept type.
   Example: *There exists exactly one accepts order activity as a part of handle order process.*

4.3.2 Reaction Rule

This rule specifies actions to be taken on the occurrence of process fact types in the state model. This rule constrains state transition of one process concept in the presence of another process fact (e.g., event being triggered).
Example: *When pending customer event triggers, It is necessary that handle customer process starts.*

4.3.3 Derivation Rule

Derivation rules are rules where element of knowledge is derived from existing concepts. SBPVR supports full derivation using if-and-only-if (equivalence) or partial derivation (using if) keywords.
Example: *Accept order activity is completed if and only if user is valid and order is accepted.*

4.3.4 Deontic Assignment

This rule expresses power, right or duty of an agent on its role in process model. Examples of this type of rules are:

1. **Authorization:** In this type of rule, commitments are assigned to agents to perform or co-ordinate work concept. This assignment depends upon the properties of work concept, properties of agents and state model.
   Examples: *It is the duty of salesman that salesman perform take order activity. It is the right of supervisor that supervisor performs reject order activity.*
2. **Event Subscription Constraint:** This type of rule constrains the agents to perceive events in the process. **Example:** *It is not permissible that a vendor perceivesaccept order event if that order is of price less than $1000.*

International Journal of Scientific & Engineering Research Volume 4, (ÚÚÜÌ㎜ÃÚ㎜)ÜÚà㎜l Ÿh Ŧ㎜
ISSN 2229-5518

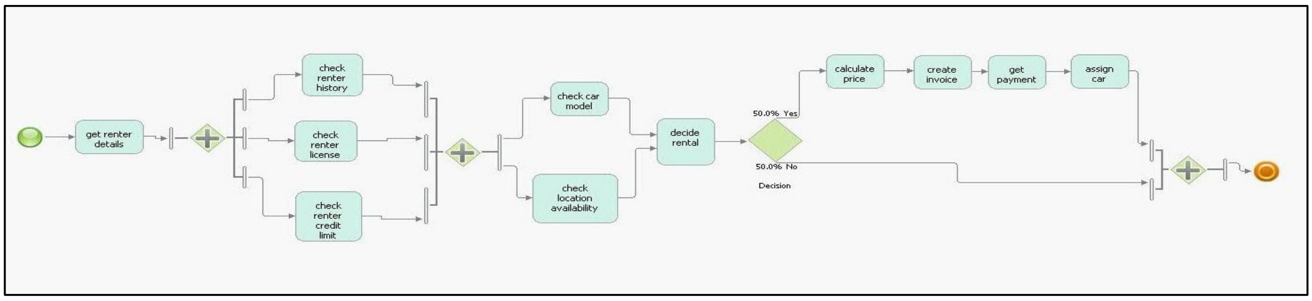

Fig. 8. Car Rental Process in BPMN

4.3.5 Execution Rule

This rule constrains or guides execution order of process elements. Although, SBPVR does not specify exact execution path of a process model, some constraints or advices to execution platform might be necessary to define at design time. For example, exclusiveness of two activities can constrain execution plan of process model. Examples for this type of rules are:

1. **Serial Order Constraint**: This rule constrains whether two activities can be executed in sequential or parallel order.
   Example: *It is not possible that interview-process and written-exam of a candidate occur at the same time.*
2. **Activity Inclusion/Exclusion Constraint**: This rule specifies inclusiveness or exclusiveness of activities. **Example**: *If reservation process includes flight-booking and reservation process includes hotel-reservation than it is necessary that reservation process includes free-carpickup process.*

5 Representation of SBPVR

5.1 Symbolization

SBPVR separates meaning of a concept from its representation. Similar to SBVR, process concepts or fact types are represented through designation. Designation for concept type is term, name or non-verbal (e.g., Icon). For fact type, designation is fact type form.

5.2 Forms Of Business Representation

Business representation also includes other forms to support meaning of concept or fact type. In SBVR, these forms are description, descriptive examples, note, comment, remark and reference. Additional forms of representation in SBPVR are:

1. **Supported Fact Types:** They for describing SBVR:fact types associated with a process concept. This representation helps in specifying relationship between SBVR vocabulary model and SBPVR process model.
2. **Process Context:** It represents context of another concept type in which it is defined.
3. **Business Goal/Purpose:** It represents business goals, tactics or strategies associated with a concept. Formal representation of these goals is out of the scope of SBPVR.
4. **NFRs:** A process concept might have non-functional requirements associated with it e.g., compliance issues, performance, security, legal issues et cetera. SBPVR does not provide formal representation for NFRs and they can be represented in natural language or using any domain specific ontology.

5.3 Example Process In SBPVR

Following is an example of Car Rental Process modeled in SBPVR. Definition of the sample Car Rental Process is taken from Knowgravity Inc. [33]. Figure 8 represents BPMN notation of the process. Section V-D and Section V-E describe SBVR and SBPVR models of the process respectively. For SBVR model, Structured English Representation (term, *verb* and keyword) [2] is used. SBPVR process concept types are represented using font sbpvr concept type. Other examples of end-to-end processes modeled in SBPVR are presented in [34].

5.4 SBVR Model

The sample process taken is a part of EU-Rent example of SBVR specification [2] which contains complete SBVR model of the same. Thus the SBVR model is not presented here and can be found at [2]. This SBVR model is used to define SBPVR elements.

5.5 SBPVR Model

Due to limitation on paper-size, only a part of SBPVR model of the Car Rental Process is presented here for reference. Complete model of this process and other sample processes are presented in [34].

- **calculate price**

Concept Type	Task type
Supported Fact Type	Salesman *calculates* rental charge Rental *has* rental charge
Supported Rules	It is necessary that the rental charge *of* each rental is calculated in the business currency *of* the rental.

- **salesman *performs* calculate price**

Concept Type	Role binding fact type

- **get payment**

Concept Type	Interaction type
Supported Fact Type	Salesman *receives* payment from renter

- **salesman *is* requesting role of get payment**

Concept Type	Role binding fact type

- renter *is* responding role of get payment

Concept Type	Role binding fact type

- get payment *transfers* payment

Concept Type	Interaction process fact type

- create invoice

Concept Type	Task type
Supported Fact Type	Salesman *creates* rental invoice

- process rental

Concept Type	Activity type

- process rental *includes* calculate price

Concept Type	Partitive process fact type

- process rental *includes* create invoice

Concept Type	Partitive process fact type

- process rental *includes* get payment

Concept Type	Partitive process fact type

- process rental *includes* assign car

Concept Type	Partitive process fact type

- sales office *coordinates* process rental

Concept Type	Role binding fact type

- **It is necessary that** check rental *starts* **only if** salesman has rental details
- **It is necessary that** process rental *starts* **only if** rental is acceptable
- **It is necessary that** create invoice *starts* **only if** rental has rental price
- **It is obligatory that** get payment *starts* **only if** rental has rental price
- **It is necessary that** assign car *starts* **only if** salesman received payment
- **It is the duty of** sales-office **that** salesman *performs* check rental

6. SBPVR To UML Activity Diagram

Not all the rules will participate in the construction of AD, as it shows the execution behavior not the structure. Ignoring the structural rules, we will consider only operative rules. A subset of those operative rules will help in generating the AD and we will name them as the automatable rules. Logical formulation of automatable rules will be used to create the activity diagram. Automatable rules are generally in the form of if-then or ECA format [35]. Logical formulation of these rules is defined as implications in SBVR. Implications will help us to find out the activities and their precondition. Looking up at all the implications until they get finished, will help us to find out the activities and their preconditions and sequence between those activities. Last implication will help us to find out the last activity and bring us to end state. A detailed discussion of this methodology is given in following sections.

6.1 Categorization Of Rules

The business rules can be categorized as structural rules and operative rules. As described in the above section, structural rules will only participate in the structure of the business organization but would not guide the business flow. Operative rules are the rules which will guide the business flow, so a distinction between the operative and structural rules is required, which has been done in the SBVR Specification [2]. As our intension is to draw the execution behavior of a system on a UML Activity diagram, we must consider only operative rules ignoring the structural rules. But not all the operative rules will participate in activity diagram, so we again categorize them according to [17] as given below:

- **IT Support is automated:** These are the rules which should be completely handled by IT system without any intervention of human user. We will refer these rules as automatable rules for rest of the discussion.
- **IT Supported is supported:** These are the rules which are supported by IT system and expect some human interventions.

The rules which are completely supported by IT will be named as automatable rules for rest of the paper, and these rules will be used to create the AD. Business rule writer is the only person who knows which rules are automatable and not. We will allow him to express this knowledge with rules at the time of writing the rules. A typical SBVR business rule signature includes some attributes whose detailed discussion is given in [2]. The 'enforcement level' attribute tells how to enforce the rule. A details list of different level of enforcement is given in [2] based on Business Motivation Model [36], but there is no level which would inform us about whether a rule is automatable or not. So, we have added another level named "automatable" which will compensate for the information required, as given in Figure 9. The result of adding the level "automatable" will result in detection of automatable and non-automatable rules at the time of writing rules. If the user sets enforcement level as automatable, it means that this rule should be completely handled by the IT system and participates in generating the activity diagram.

It is necessary that create invoice *starts* only if rental *has* rental price

Guidance Type:	Operative Business Rule
Description:	An invoice will be created when rental is charged with a rental price
Enforcement Level:	automatable
Supporting Fact type:	**create invoice *starts*, rental *has* rental price**

Fig. 9. Rule Signature for automatable rule

6.2 Logical Formulation Of Automatable Rules

Automatable rules generally show the execution of activities and most of the activities should be guarded by some preconditions. This is why most of the automatable rules exist in if-then construct in SBPVR, but they may exist in some more constructs. For this paper, we are mainly handling if-then construct. In SBPVR, the logical formulation of if-then rules is given as "Implication". An if-then rule relates the activity and its preconditions with the consequent and antecedent of the implications, respectively, as shown in the Figure 10. To deduce the "if p then q else r", we will decompose the else part again in "if-then" construct. For example

if p then q else r

can be decomposed as

if p then q and if !p then r

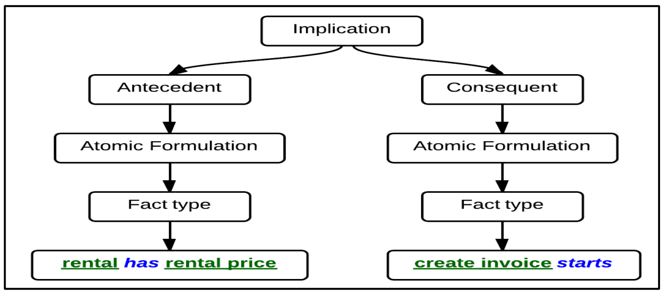

Fig. 10. Logical formulation of if-then rules

6.3 Fact Types: Activities In An Activity Diagram

As we have already discussed that a fact type can be represented as term-verb-term. A verb can be of two types, one which imply some action (transitive verbs) and other which doesn't imply any action (intransitive verbs). The statements involving the transitive verbs will represent some action. For example, the fact type "create invoice starts" is a statement which include a transitive verb "starts" and shows an action "start of invoice". These types of fact types can be assumed as the activities. The fact types having intransitive verb e.g. 'has' or 'is of' will not be considered as the activities because they show the structural nature not the imperative nature. For example, "rental has rental price" shows a possession not an activity.

Naming of activities can be done in the following two ways:

1. **Direct Naming:** Use the fact type directly as the activity name.
2. **Objectification:** Use the objectification of fact types. SBVR has a provision for giving instances of fact types a type name; it is called "objectification" [2]. For example, an occurrence of the fact type "salesman receives payment" might be objectified as an "receive payment". An objectification allows us to predicate things about a fact type, like how, when and where they happened, etc. Each activity in a UML AD may correspond to an objectification. The corresponding transformation turns the fact type into a command to bring such an action into existence. Use of the infinitival form of the verb phrase of the underlying fact type for an activity name might be preferred, to reflect the imperative nature of the activity: "receive payment," the successful outcome of which is an "payment received", which is a state of affairs (event) that salesman has received a payment.

The fact types associated with the automatable rules will be used to form an ordered set of commands. It is a very sensitive and non-trivial process as it must consider both linguistic and logic especially the temporal aspects all of which may not be specified directly in the set of declarative rules. This transformation must be logically consistent with the conceptual schema of SBVR.

6.4 SBVR To UML Activity Diagram Mapping Rules

This section mainly deals with the mapping of SBPVR to UML Activity diagram.

6.4.1 Initial Node:

This is the start state of the activity diagram. It doesn't play a very important role but significantly shows the starting point of a scenario. We have given it a default name "start".

6.4.2 Activity Node:

As we have already discussed, the fact types having transitive verbs will be assumed as the activity node.

6.4.3 Activity Edge:

An activity edge is a set of events, guard conditions and actions which allows the transition from one activity node to another activity node. An event is the trigger of the transition. Upon triggering the transition, the condition is checked, if the condition holds true, then the corresponding action occurs and brings another activity into existence. It is not necessary that a transition must have trigger. Transitions without the trigger are known as trigger less transitions. A typical SBPVR operative rule may be written as:

upon event, if <propositional expression 2>, then <propositional expression 1>.

But we are using the following format of operative business rules in SBPVR

<propositional expression 1>[if<propositional expression 2>]

So, the mapping from these types of SBVR rules to UML AD will create the trigger less transitions. The propositional expression 2 will help to find out the guard condition and propositional expression 1 will help to find out the action.

- **Guard Condition:** Assume an operative business rule like given below:
  It is necessary that assign car *starts* only if salesman received payment
  The propositional expression 2 in if clause refers to the fact type 'salesman received payment'. Depending upon the fact type, we are creating a boolean variable like 'received payment'. And the guard condition will become 'received payment == true' as shown in Table I.

Table I
Fact Types (Activities) And their Pre-Conditions

Fact Type	**Boolean Variable**	**Pre-condition**
salesman received payment	received_payment	received_payment = true
assign car start	car_starts	car_starts=true

- **Action:** These are the actions which should be invoked during the transition from one activity to another. For example in the above rule, if a salesman received payment, then the "rented car start". What we are doing is taking the propositional expression 1 and find out the corresponding fact type. Merging of verb and last term of the fact type will create the name of the action like "car starts()" as shown in Table II.

Table II
Fact Types (Activities) and corresponding Actions

Fact Types	**Corresponding Actions**
salesman received payment	payment_received()
assign car start	car_starts()

6.4.4 Fork/Join Node:

- **Fork Node:** This is a pseudo-state where one transition is coming and multiple parallel transitions are going out of it. A SBVR rule like "it is necessary that assign car starts and travel time starts only if salesman received payment" represents the enforcement of two activities "assign car starts" and "travel time starts" when salesman received the payment. This situation will generate the fork state. There will be an incoming transition having the guard condition "received payment=true" and two outgoing parallel transitions pointing toward activities "assign car starts" and "travel time starts".
- **Join Node:** This is another pseudo-state where multiple parallel transitions are coming and only one transition is outgoing. The generation of this state will be same as the above except there will be multiple guard conditions and only one outgoing transition.

6.4.5 Activity Group

The ActivityGroup in UML activity diagrams are generally known as swimlanes which basically represents who is doing the activity. The entity doing an activity will be referred as the giver of the activity. As we have already discussed that the representation of fact type (activity) can be of two type, active form and passive form. In a sentence having an active, transitive verb, the giver of the action of the verb is the subject of the sentence. In English, the "giver" corresponds to the object filling the role of the first placeholder in the fact type form, e.g. "customer" in "customer places order." If the fact type form is passive, e.g. "order is placed by customer," it is the reverse. These mean the same thing. They are synonymous forms. Facts of either of these forms would be logically equivalent.

6.4.6 Activity Final Node

This is the point in an activity diagram where all the activities get end up. We are creating a default end state with the default name "End".

7. Rule Sequencing Engine

Rule Sequencing Engine (RSE) is an engine used to establish the order between activities. The engine consist of a data structure (RSE-DB) used to contain the guard conditions which have been occurred as true and a decision unit to decide the next activity. For example, if we encounter a rule like below, then the fact type corresponding to if clause is "salesman received payment" and the corresponding guard condition becomes "received payment = true".

It is necessary that assign car *starts* only if salesman received payment

The RSE search its RSE-DB for this condition to be true, if the condition exist and holds true, then the fact type 'assign car starts' corresponding to then clause will be the next activity. After getting the next activity, a boolean variable 'car starts' is created and get set to true and inserted into the database. This concept is motivated from OMG's Production Rule Representation (PRR) [37] and RETE Algorithm.

8. Algorithm

The vocabulary and rules are parsed that instantiate the SBVR Meta-Model given in [2]. Chapter 9 in [2] deduces the logical formulation of a SBVR business rule. The logical formulation of an automatable operative business rules will be of our interest at the moment, as we are modeling the activity diagram. Since we are generating the UML AD based on the business artifacts given only in if-then rules, there must be exactly one automatable rule which would not have any if clause. This is because the absence of pre-condition will allow the activity to occur initially. If there is more than one such rule, then there will be two transitions from start state which is against the UML AD Semantics. It is only possible if those two activities occur in parallel, which will be modeled as fork in AD. The fact type corresponding to the automatable rule having no if clause will be the first activity. Create a boolean variable as shown in the section 6.4.3, assign it as true and put into the RSE-DB. The detection of further activities will be done with the help of "implications" given in SBVR Metamodel.

After deducing the logical formulations, we will look up at all the 'obligation claims' [2] that are 'implications' to find out the relation between the fact types of 'if' construct (antecedent of the implication) and those of 'then' construct (consequent) in an if-then construct. To find out the next activity, check the RSE-DB to known whether the guard condition corresponding to the fact type in antecedent is true or not. If the guard condition holds, find out the fact type corresponding to consequent of the implication and make it as the next activity and also create a corresponding Boolean variable with assigning "true" value and put it into the database of RSE. Also, create a transition from previous activity to the current activity as explained in section 6.4.3. If the guard condition doesn't holds, then search for the next implication until all the implications get visited. The absence of such an implication will result in the end state.

The flow chart shown in Figure 11 presents a general algorithm to find out the activity diagrams. The special situations like an activity doesn't have any outgoing transition, fork and join; multiple incoming transitions to the end state are not shown in the flow chart. They can be directly hard coded on top of the basic algorithm shown in above flow chart.

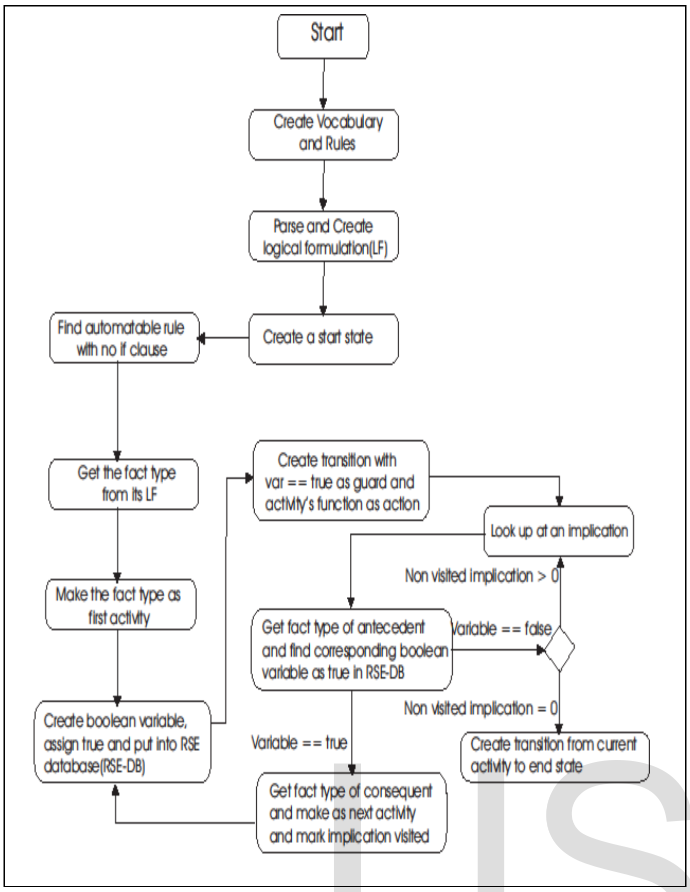
Start
Create Vocabulary and Rules
Parse and Create logical formulation(LF)
Create a start state
Find automatable rule with no If clause
Get the fact type from its LF
Make the fact type as first activity
Create boolean variable, assign true and put into RSE database(RSE-DB)
Create transition with var == true as guard and activity's function as action
Look up at an implication
Non visited implication > 0
Get fact type of antecedent and find corresponding boolean variable as true in RSE-DB
Variable == false
Non visited implication = 0
Variable == true
Create transition from current activity to end state
Get fact type of consequent and make as next activity and mark implication visited

Fig. 11. Flow Chart of SBVR To UML Activity Diagram Transformation

8. SBPVR To UML Sequence Diagram

The scenarios within an environment can be represented as a sequence diagram which is described in [38]. Whittle et al. [38] illustrate that a sequence diagram can be represented as the set of messages with the information of their source and destination object in a sequential manner which is represented below:

$src_1 \rightarrow msg_1 \rightarrow dest_1$; $src_2 \rightarrow msg_2 \rightarrow dest_2$; ….. $src_n \rightarrow msg_n \rightarrow dest_n$;

To draw the sequence diagrams for the scenarios described by SBPVR Rules, we will use the AD generated in the last section.

8.1. Mapping SBPVR To UML Sequence Diagram Elements

The SDs requires the messages having proper sequence between them along with their source and destination object information. The AD contains the activities with proper sequence but don't give any information about the objects. In the following sections, we will see how the activities and their sequence in AD can be used to create SDs.

8.1.1 Message: The messages in a SD are responsible for the occurrence of events and actions in the sequence diagram. Due to its logical similarity with the activities of AD, they can be given the same name as of the activities. For example, the activity "salesman received payment" in an AD can be a message in a SD. The messages toward the life line of an object show the event on the object while the messages away from an object are considered as the actions of that object.

8.1.2 Source Object Of A Message

The source object of a message will be the 'doer' of activity having same name as of that message. The semantics of 'doer' is same as the 'subject' in an English sentence having an active verb. The subject is the noun who performs the action of the verb. In SBVR's structured English, a sentence having an active, transitive verb, the 'doer' of the action of the verb is the object filling the role of the first placeholder [2] in the fact type form e.g. "salesman" in "salesman receives payment". If the fact type form is passive e.g. "payment received by salesman", then the ′doer′, would be the object filling the role of last placeholder. The above two sentences have the same meaning and must be presented as the synonym of each other by BA during the development of Vocabulary. However, it is preferable to use the active form wherever possible. The above two fact type forms would have the same logical formulation or logically equivalent.

8.1.3 Destination Object Of A Message

The destination object of a message is one at which a message gets end up. That object will be the active object in the system means the execution control will retain with this object only. Every message has its own source and destination. For example take two messages as shown below:

$source_1 \rightarrow message_1 \rightarrow destination_1$
$source_2 \rightarrow message_2 \rightarrow destination_2$

Assume message2 is next to message1 in the sequence. After the occurrence of message1, destination1 will be the active object and for the occurrence of message2, source2 should be the current active object. This implies that destination1 should be same as source2 as shown below. It implies that the destination of a message will be the source of the next message in the sequence.

$source_1 \rightarrow message_1 \rightarrow destination_1 = source_2 \rightarrow message_2$

8.1.4 State Invariant

An object in its life-time, passes through several states. These states are the different configuration of variables of the object. For example, the object "salesman" has the variables 'renal details' and 'payment' of type string where "renal details=d and payment=null" represents a state of "salesman". The state change of the object occurs due to their action and events. For example, the action "salesman receives payment" will change the value of 'payment' variable as "payment=p" and results in a new state (new configuration) of salesman "renal details=d and payment=p". The invariants of the state can be attached in many ways like in plain English or Object Constraint Language (OCL) [39]. It depends upon the user and his requirements, how to attach them.

8.1.5 General Ordering

The general ordering in UML SD Metamodel [40] is a partial ordering between the two messages. So, we will look up only at the two activities in UML AD generated above and map their ordering to the general ordering of messages.

International Journal of Scientific & Engineering Research Volume 4, [illegible]
ISSN 2229-5518

8.2 Algorithm

The generation of UML SD involves the collaboration of both UML AD and SBVR Metamodel. The flow chart for this transformation is shown in Figure 12. The algorithm tries to find out the messages of the SD. According to [41], only activities of the AD can be transformed to messages of the SD. Hence, we will set all the activities of AD as the messages of SD. The first activity in AD will be mapped to the initial message of SD. And the action corresponding to this initial message will be the same as the action corresponding to that activity in AD. This action will become the send event of the destination message end. We will generate the general ordering of messages through the sequence between the activities in AD. The next thing is to find out source and destination life lines for a message. A detailed discussion of algorithm is given below.

The name of a message would be the name of activities which in turn is the fact types. In SBVR metamodel, each fact type has some roles which are situated at some placeholder. If the fact type (which is a message in SD) is in active form, the source life line of the message will be the term at first placeholder else it will be the term at last placeholder. To find out the destination life line, we will look up at the next message of current message in general ordering. The source of the next message will be the destination life line of current message as described in section 8.1.3. This whole process is repeated again from finding the next activity, creating a message and finding the source and destination lifelines for messages, until we will not traverse all the activities in AD and reach the end state from all the possible paths. There may be some consecutive activities in AD whose sources are same. For example the consecutive activities "salesman receives userdetails" and "salesman receives payment" have the same source 'salesman'. In this case, the message corresponding to the activity "salesman receives userdetails" will be a self message. A self message is a message whose source and destination are same. Due to this self message, the active object after the occurrence of this message will remain the same which is 'salesman'. And next message "salesman receives payment" will be sent from the object 'salesman'. It may also happen that the activity next to current activity doesn't exist in an AD, for instance, the activity is the immediate previous to the end state. In this case, this activity does not have any next activity. We may have to compromise as there is no information for the next object. If we see the messages, they are transferring the control to the next destination object. And if there is no information about the next active object to take over the control, we have to keep the control to current object. Due to which, the destination life-line for this message will be the same as source life line. This is a limitation of this approach. In this approach, we are missing the detection of actors. As our intension is to map the SBPVR Metamodel to UML SD metamodel not the UML SD syntax, this is not important at the moment as UML SD Metamodel doesn't include any entity like actors.

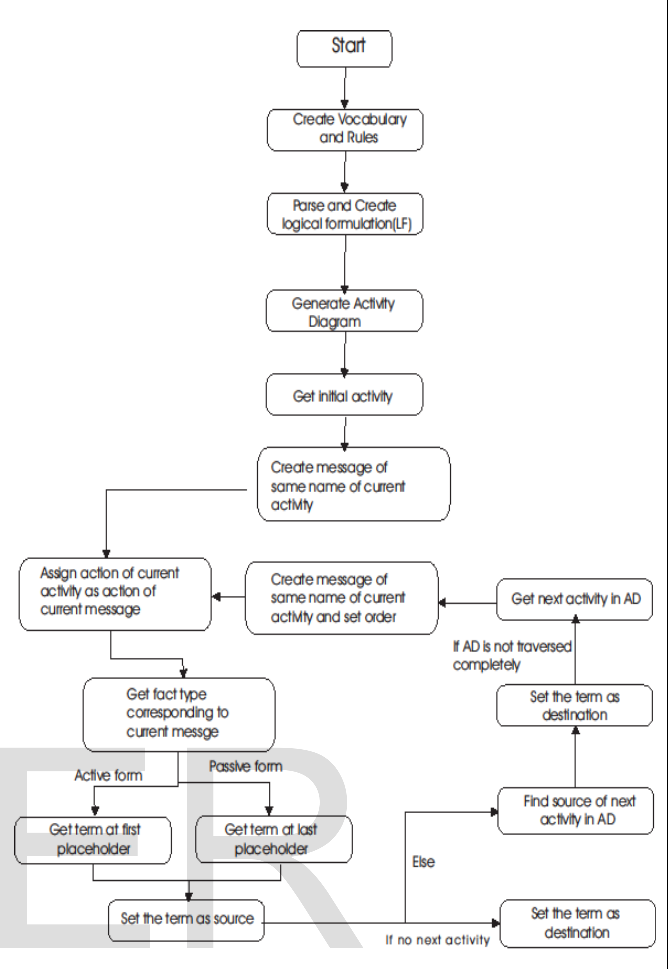
Start
Create Vocabulary and Rules
Parse and Create logical formulation(LF)
Generate Activity Diagram
Get initial activity
Create message of same name of current activity
Assign action of current activity as action of current message
Create message of same name of current activity and set order
Get next activity in AD
If AD is not traversed completely
Get fact type corresponding to current messge
Set the term as destination
Active form
Passive form
Find source of next activity in AD
Get term at first placeholder
Get term at last placeholder
Else
Set the term as source
If no next activity
Set the term as destination

Fig. 12. Flow Chart of SBVR To UML Sequence Diagram Transformation

9. SBPVR To UML Class Diagram

The SBPVR statements are declarative in nature. These statements are used to declare the structural and operative behavior of a system. The structure of the system mainly involves the classes, attributes, functions and relationships between those classes. For example, 'salesman' is a class that is a SBPVR term and 'salesman has rental details' is a SBPVR fact type that shows the class 'salesman' has an attribute of type 'rental details'. The business rules determine the correct value of the properties that an object will have and methods of deriving the information needed by a class. Appendix H in SBVR Specification [2] gives a mapping from the SBVR Vocabulary and Rules to the CD but it is not sufficient as it does not give any information about the functions of business objects. The contribution of this paper is a mechanism to find out the functions of the classes, association between them and cardinalities for association ends. Some of the important rules of the mapping are described here.

- **Class Name:** These are the nouns or group words which are used to define a concept and starts with small letter. They are represented as classes in UML CD.

- **Instance Name:** The individual concepts behave as an instance of a particular class. The name is followed by a colon and then by the term for its general concept [2]. E.g. The Name 'John' is an instance of the class 'salesman' as shown in Figure 13.

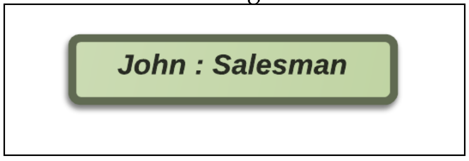

Fig. 13. Class and its instance name

- **Attributes:** For the binary fact types [2] that have 'has' as the conjunction, the term at the last placeholder will be represented as an attribute of the class if the sentence is the active sentence else it will be the term at first placeholder. For example, in a binary fact type in active form 'salesman has rental details', the class 'salesman' will have an attribute of type 'rental details'. In general, a unary fact type is transformed into a UML Boolean attribute. For example, in the unary fact type 'rental car starts', the class 'rental car' will have a boolean attribute 'isStarted' as shown in Figure 14.

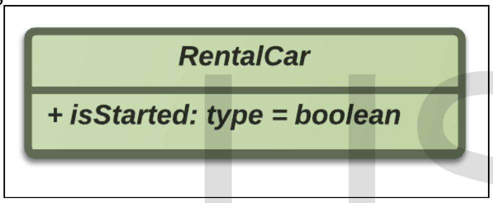

Fig. 14. Class and its instance name

- **Generalization/Specialization:** The SBVR Specification says that 'concept1 specializes concept2' and 'concpet1 generalizes concept2' that means a term can be specialized form or a generalized form of another concept. For example, in SBPVR, there is a term 'salesman' that is a role of 'Employee'. In this example, the term "salesman" is a specialized role of 'Em ployee'. So, the class corresponding to the term 'salesman' will be a subclass of the class corresponding to the term 'Employee' as shown in Figure 15.

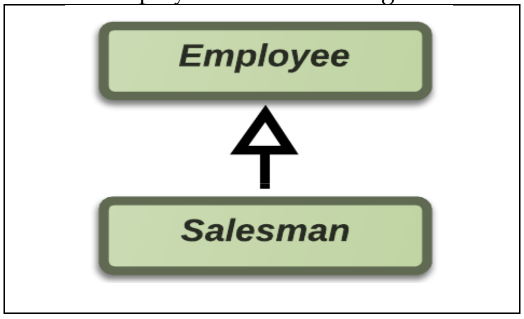

Fig. 15. SuperClass and SubClass.

- **Functions:** SBVR specification helps us to construct the CD but does not give any information about functions of classes. Before we draw the class diagram, we should know which function belongs to which class. To find out this information, we will follow the same approach as we followed to generate the SD. In a SD, one object interacts with another by sending it a message. For instance, the 'salesman' sends a message "salesman received payment". In the sequence diagram, this message is associated with an action "receive payment()". Whittle et al., [38] has suggested a relationship between SD and CD that help us to conclude that the action corresponding to the messages will become a function for the source object that is 'salesman'. So, the class 'salesman' will have the function "receive payment()". The paper is not able to find out the return type and arguments of these functions at the moment. We will recommend the user to set these entities manually.
- **Association:** The association between the classes can be figured out from fact types. Binary Fact types [2] actually establish the relationship between the two business entities. For example, the fact type "salesman receives payment" establishes the relationship between the salesman and payment.
- **Multiplicity:** There are some constraint rules which constraint an association between the two classes. In these rules, the quantifier associated with a term tells about the cardinality at association ends. For example, the following rules says that each customer rents at most one car. The quantifier 'at most one' tells that the multiplicity at 'car' end should be '0..1' as shown in Figure 16. This concept is motivated from [18].

It is necessary that each customer *rents* at most one car.
It is necessary that each customer *contacts* exactly one rental office.
It is necessary that each rental office *employ* at least one saleman.

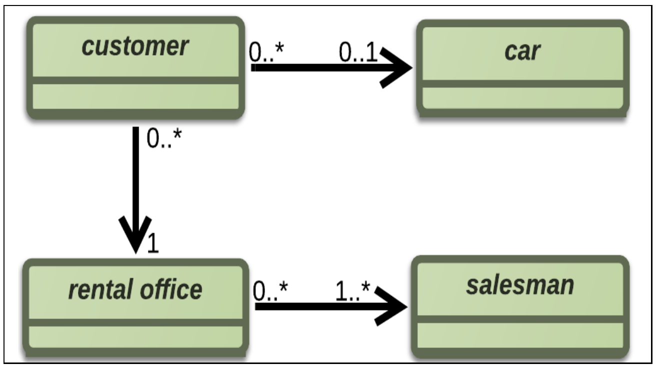

Fig. 16. Multiplicity and association between classes

10. Conclusion

In this work, we have made an initial attempt to define a generic meta-model named, SBPVR, for declarative business process modeling. SBPVR follows SBVR's fact oriented approach and process models in SBPVR are built over SBVR vocabulary & rule models. SBPVR categorizes process knowledge into process concept types, process fact types and process rules. We have extended these categories further, to represent various process elements and their semantics. Benefits of the declarative nature of SBPVR are flexibility and adaptability which are critical requirements for knowledge intensive and dynamic process models. SBPVR enables integration between rule model and process model, which covers maximum representational constructs using uniform methodology.

We have presented a methodology to generate UML AD, SD and CD from the SBPVR model driven business process. These business processes are CIMs and UML models are PIMs in MDA. SBPVR allows developing the business process vocabularies which include the basic business terms and fact type, and business rules. The work in the paper basically bridges the gap between business modeling people and IT people. The business people who are interested in writing the business processes will write them in SBPVR that needs some external efforts to extract the imperative nature embedded in those declarative sentences. This imperative nature can be shown in UML and BPMN too. We have chosen UML because it is more efficient and adaptable for example, a UML AD can be used as a workflow specification language and also they are very efficient for reverse and forward engineering. Main contribution of this paper is to bridge the gap between business people and IT people by allowing them to convert business processes into platform independent UML AD, SD and CD, which can be further transformed to application code to check any inconsistencies between the actual and intended behavior.

REFERENCES

[1] R. G. Ross, Principles of the Business Rule Approach. Addison-Wesley Information Technology Series, 2003.
[2] "Semantics of business vocabulary and rules specification," Object Management Group, March 2006.
[3] M. Z. Muehlen, M. Indulska, and G. Kamp, "Business process and business rule modeling: A representational analysis," in EDOCW '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 189–196.
[4] O. M. Group, "Business Process Modeling Notation (BPMN)," January 2008, http://www.omg.org/spec/BPMN/1.1/.
[5] OASIS, "Web Services Business Process Execution Language (WSBPEL)," http://www.oasis-open.org/home/index.php/.
[6] A.-W. W. Scheer, Aris–Business Process Modeling. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1998.
[7] W. M. Van Der Aalst and A. H. M. Ter Hofstede, "Verification of workflow task structures: A petri-net-based approach," Inf. Syst., vol. 25, no. 1, pp. 43–69, 2000.
[8] O. M. Group, "Unified Modeling Language: Activity Diagram," http://www.uml.org/.
[9] W. M. P. van der Aalst and M. Weske, "Case handling: a new paradigm for business process support," Data Knowl. Eng., vol. 53, no. 2, pp. 129–162, 2005.
[10] S. Goedertier and J. Vanthienen, "Designing compliant business processes with obligations and permissions," Business Process Management Workshops, pp. 5–14, 2006.
[11] M. Pesic and W. van der Aalst, "A declarative approach for flexible business processes management," 2006, pp. 169–180. [Online]. Available: http://dx.doi.org/10.1007/11837862 18
[12] W. M. P. van der Aalst and M. Pesic, "Decserflow: Towards a truly declarative service flow language," in The Role of Business Processes in Service Oriented Architectures, 2006.
[13] S. W. Sadiq, M. E. Orlowska, and W. Sadiq, "Specification and validation of process constraints for flexible workflows," Inf. Syst., vol. 30, no. 5, pp. 349–378, 2005.
[14] A. Raj, T. V. Prabhakar, and S. Hendryx, "Transformation of sbvr business design to uml models," in Proceedings of the 1st India software engineering conference, ser. ISEC '08. New York, NY, USA: ACM, 2008, pp. 29–38. [Online]. Available: http://doi.acm.org/10.1145/1342211.1342221
[15] A. Agrawal, "Semantics of business process vocabulary and process rules," in Proceedings of the 4th India Software Engineering Conference, ser. ISEC '11. New York, NY, USA: ACM, 2011, pp. 61–68. [Online]. Available: http://doi.acm.org/10.1145/1953355.1953363
[16] M. H. Linehan, "Semantics in model-driven business design," Models/UML Conference, 2001.
[17] M. Schacher, "Moving from zachman row 2 to zachman row 3," Business Rules Journal, vol. 7, no. 6, June 2006.
[18] "Executable uml specification," Object Management Group, May 2004.
[19] H.-E. Eriksson and M. Penker, Business Modeling with UML: Business Patterns at Work. New York: John Wiley and Sons, Inc., 2000.
[20] P. Musham, S. Singh, R. Bahal, and P. Tv, "Visual SBVR," in Digital Information Management, 2008. ICDIM 2008. Third International Conference on, November 2008, pp. 676–683.
[21] D. Sorensen, A. Pastiak, A. Mitra, and A. Gupta, "Integrating ontology into sbvr," Eller College of Management, Report 1033, 2006, uRL: http://www.knowgravity.com/pdf-e/CASSANDRA xUML E.pdf.
[22] D. T. Maurizio and C. Pierpaolo, "Sbeaver business modeler editor," uRL: http://sbeaver.sourceforge.net/.
[23] "Digital business ecosystem project, "an internet based software environment in which business applications can be developed and used"," Project, uRL: http://www.digital-ecosystem.org/.
[24] A. G. Kleppe, J. Warmer, and W. Bast, "Mda explained: The model driven architecture: Practice and promise," Boston, MA, USA, 2003.
[25] omg, Meta Object Facility (MOF) Core Specification Version 2.0, 2006. [Online]. Available: http://www.omg.org/cgi-bin/doc?formal/2006-01-01
[26] C. Schlenoff, G. M., F. Tissot, J. Valois, J. Lubell, and J. Lee, "The process specification language (PSL): Overview and version 1.0 specification," Gaithersburg, MD., 2000, nISTIR 6459, National Institute of Standards and Technology.
[27] J. Clark, C. Casanave, K. Kanaskie, B. Harvey, N. Smith, J. Yunker, and K. Riemer, "ebXML business process specification schema version 1.01," 2001.
[28] K. Taveter and G. Wagner, "Agent-oriented enterprise modeling based on business rules," in ER '01: Proceedings of the 20th International Conference on Conceptual Modeling. London, UK: Springer-Verlag, 2001, pp. 527–540.
[29] A. K. Chopra and M. P. Singh, "Commitments for flexible business processes," in AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems. Washington, DC, USA: IEEE Computer Society, 2004, pp. 1362–1363.
[30] D. Hay, "Defining business rules; what are they really?. guide business rules project, final report," 2000.
[31] S. Goedertier, R. Haesen, and J. Vanthienen, "EM-BrA2CE v0.1: A Vocabulary and Execution Model for Declarative Business Process Modeling," SSRN eLibrary, 2007.
[32] S. Jablonski and C. Bussler, Workflow Management: Modeling Concepts, Architecture and Implementation. International Thomson Computer Press, September 1996.
[33] M. Schacher, "Eu-rent business model, know gravity inc." http://www.knowgravity.com.
[34] A. Agrawal, "Semantics of Business Process Vocabulary and Process Rules and a Visual Editor of SBVR," Master's thesis, Indian Institute of

Technology Kanpur, India, 2009, http://www.cse.iitk.ac.in/users/agrawala/thesis.pdf.
[35] F. Bry and P. lavinia Patranjan, “Use cases for reactivity on web: Using eca rules for business process modeling,” INSTITUT FUR INFORMATIK der Ludwig-Maximilian-Universitat Munchen, Report, 2006, uRL: http://www.pms.ifi.lmu.de/publikationen/diplomarbeiten/Inna.Romanenko/DA Inna.Romanenko.pdf.
[36] “Business motivation model,” uRL: http://www.omg.org/docs/dtc/06-08-03.pdf.
[37] “Production rules representation,” Object Management Group, Speci ication, 2005, uRL: http://www.w3.org/2004/12/rulesws/ slides/paulvincent.pdf.
[38] J. Whittle and J. Schumann, “Generating statechart designs from scenarios,” International Conference on Software Engineering, pp. 314 – 323, 2000.
[39] “Object constraint language specification,” 2003, uRL: http://www.omg.org/docs/ptc/03-10-14.pdf.
[40] J. Rumbaugh, I. Jacobson, and G. Booch, Unified Modeling Language Reference Manual, The (2nd Edition). Pearson Higher Education, 2004.
[41] P. Selonen, K. Koskimies, and M. Sakkinen, “Transformation between uml diagrams,” Journal of Database Management, vol. 14, no. 3, pp. 37–55, 2003.