ON EXPRESSING BUSINESS RULES WITH A COMBINATION OF UML AND OCL*

Kestutis Normantas, Olegas Vasilecas

Vilnius Gediminas Technical University, Information Systems Research Laboratory, Sauletekio al. 11, LT20221, Vilnius, Lithuania, {kestutis.normantas,olegas,sergejus}@isl.vgtu.lt

Abstract. Today’s volatile, competitive business environment demands flexible software systems that could fit changing business requirements. Therefore business rules are of great importance in the development of software systems. The Object Constraint Language (OCL) as part of the Unified Modeling Language (UML) provides the possibility to express business rules in a conceptual model unambiguously. In this paper we analyze possibilities to express different kinds of business rules with UML/OCL. We show how UML/OCL can be used in business application constraint modelling and discuss their advantages and limitations.

Keywords: Business rules, UML, OCL

1 Introduction

Modelling business rules (BR) is pivotal for analysing and designing information systems that are in line with company’s long term strategy and that efficiently support its core business processes. According to [1], BR as their predecessors (business scope, motivation and strategy) and as their successors(BR model and executable code) are concurrent with other aspects of enterprise system from different points of view. Therefore, integration of BR model with other system’s models as well as clear and unambiguous understating of BR and possibility to access and manage them in-time is crucial.

An enterprise operates according to many different kinds of rules, such as legal mandates and rules it constructs for itself [2]. The most basic element of a BR is the language used to express it [3]. The most understandable form of BR is a natural language. However, this form is ambiguous and informal to use it in information systems development process. One way to express BR formally in object-oriented software systems design is the Object Constraint Language (OCL). OCL is a part of the Unified Modeling Language (UML) [4], compatible with the leading object-oriented software development methods. OCL is a formal language, similar to structured English, to express side-effect free constraints within UML model [4]. In UML 1.1, the main purpose of OCL was to indicate the constraints on model elements in diagrams. UML 2.0 has expanded the role of OCL; thereby recent OCL 2.0 version is fully integrated with UML metamodel, providing possibility to define queries, reference values, or state conditions and BR in a model. Additionally, current UML/OCL modelling tools support generation of OCL expressions to executable code; consequently, system modellers are able to build an application by creating a platform independent UML/OCL model and transforming it to platform dependent code.

Despite that OCL application domain is very wide and there are presented many researches regarding different aspects of OCL employment in systems modelling [5], [6], [7], [8], comprehensive research considering to modelling of BR is not presented in BR research area. Erricson and Penker in [9] demonstrated all possibilities to model a business with UML with a facile explanation how to describe business rules using OCL 1.0 syntax. However, they discussed on it considering a very basic classification of BR; moreover, they did not educe advantages and disadvantages of OCL as the language for expressing of BR.

More extensive research is presented by Sosunovas and Vasilecas in [10], where research on modelling of different kinds of BR was presented. Nonetheless, investigation is made using the first version of OCL, therefore analysis of expressiveness of the OCL was limited due to limitation of the OCL syntax. Warmer and Klappe in [11] described how to use the second version of OCL in business modelling by providing a wide range of examples; however, they discussed on it from the OCL point of view, therefore particularity of BR was not examined.

In this paper we present investigation on expressiveness of OCL 2.0 to capture different kinds of BR. As BR classification schema we chose extended list of action assertion types presented by GUIDE project [3], because it in to a certain extent involves other classifications presented by well-known researchers Ross [13], Morgan [14], and von Halle [2].

This paper starts by giving a brief overview of OCL in Section 2.1. Then in Section 2.2, it gives an explanation of selected example model. Next, in Sections 2.3-2.10, it investigates on expressiveness of OCL by applying it to a selected model. Finally in Section 3, the paper concludes presented research.

* The work is supported by Lithuanian State Science and Studies Foundation according to High Technology Development Program Project “Business Rules Solutions for Information Systems Development (VeTIS)” Reg. No. B-07042
2 Expressing business rules with Object Constraint Language

2.1 The Object Constraint Language

The Object Constraint Language is a formal language allowing the specification of constraints in context of UML model. According to [4], OCL developed as business modelling language and has its roots in the Syntropy method. It has been adjusted to UML to compliment UML modelling, because UML is not expressive enough to provide all the relevant details of a specification. However, OCL is side-effects free language; therefore, it cannot change anything in the model. The state of a system will never change because of the evaluation of an OCL expression, even though an OCL expression can be used to specify a state change (e.g., in a post-condition) [11]. Furthermore, it is not possible to write program logic or flow control in OCL as well as express implementation details.

The main characteristics of OCL 2.0 are [11]:

- Both query and constraint language – it is possible to write a query expression of a body of an operation as well as define constraint to some attribute’s value or existence of some object.
- Mathematical foundation – it is based on mathematical set theory and predicate logic, and it has a formal mathematical semantics.
- Strongly typed language – model elements used in OCL expressions must conform by types, therefore OCL expressions can be checked during modelling.
- Declarative language – modeller can make decisions at a high level of abstraction, without going into detail how something should be calculated.

The main purpose of OCL is [11]:

- specify invariants on classes, types or stereotypes in class model,
- specify pre and post conditions of operations,
- describe guards on transitions in state diagrams,
- specify target for messages and actions,
- specify derivation rules for any expression over a UML model.

Considering OCL characteristics and using UML 2.0 in combination with OCL 2.0 different types of business rules will be discussed in next subsections. Hence, we assume reader is conversant to above-mentioned techniques. For system specification we used standard UML 2.0 notation, which description and modelling guidelines could be found in [9].

2.2 Example Model

As an example, we use simplified document management system specification based on model requirements for electronic document and records management systems Moreq2 [15]. Related class diagram presented in Figure 1.

In example model document must have a type, a kind and some priority. Every document must be created by some author which may be a user or a group as well as some contact or a contact group. Document kind of receivable must be received by some addresser and kind of outgoing must be sending to some addressee. According to processes with documents, document may be in some state. Some documents may be involved in a file, which is virtual catalogue for collecting documents.

Presented example is composed of different BR. For example document type may be one of enumerated document types: letter, invoice, memorandum, etc. To collect and examine different BR classification schema is required. To understand the nature of BR and the categories into which they fall, classification schema is presented in GUIDE Business Rules Project report [3]. Though, there are a lot of proposals how describe and classify BR in business rules research community, however GUIDE presents a formal approach for identifying and articulating the rules that define the structure and control the operation of an enterprise [3], moreover classification proposed in this project involves others, presented in [2],[13],[14].
Considering extended list of action assertion type rules described in GUIDE report, expressiveness of OCL to describe BR will be investigated in next subsections. Action assertion types involves following types: instance verifiers, type verifiers, sequence verifiers, position verifiers, functional evaluators, comparative evaluators, calculators, update controllers and timing controllers.

2.3 **Expressing instance verifiers**

Each rule describes the effect of a correspondent (constraining object) upon an anchor (constrained object) [13]. Instance verifiers pertain to individual instances or occurrences of correspondent object classes [3]. Instance verifiers type includes following subtypes: mandatory constraint (requiring occurrence of some object), limited constraint (constraining number of population of object), restricted (involving recursive structures), pre-existing (requiring occurrence of some correspondent object class to exist before anchor object class and existence at the moment of rule check) and antecedent (requiring occurrence of correspondent object class to exist before anchor object class and it is not important or it still exists during check of rule).

Instance verifiers require possibility to manipulate with a population of corresponded object class. The manipulation of collections of objects is common in object-oriented systems; thus, OCL support different operations on collections. All operations on collections are denoted in OCL expressions using an arrow; the operation following the arrow is applied to the collection before the arrow [11].

Though, the most rules of this type could be modelled using standard UML notation, e.g. mandatory or limiting constraints as cardinality of association between classes, or cardinality of object attribute, there is possibility to express them using OCL. For example, the rule stating that „Every document must have a responsible user assigned” can be expressed as cardinality of attribute (or association) „responsible”, or as following OCL expression:

```
context: Document
inv: self.responsible->notEmpty()
```

OCL expression above is stated as invariant in context of document. An invariant is a constraint that should be true for an object during its complete lifetime [11]. Thence, this rule states while document exists it
must have responsible user assigned to it. In the same manner could be modelled limited constraint, e.g. “A file must not have more than 1000 document involved in”:

\[
\text{context: File} \\
\text{inv: self.consistOf->size()} < 1000
\]

The invariant above states that the size of the collection (number of all elements) of involved in file document objects must be less than or equal to 1000. Operation on collection size() is used to indicate the size of collection. Similar operation count() could be used to check the number of occurrences of some object in the related collection.

For restriction of recursive structures, the recursive associations should be introduced in UML model. In the considering example, recursive associations are modelled as document class attributes “isBasedOn” and “isLinkedWith”. Suppose rule stating that “A document cannot be based on itself”. In this case operation reject(), could be involved:

\[
\text{context: Document} \\
\text{inv: self.isBasedOn->reject(self)}
\]

Operation reject() in this expression is used to state that in associated with document object collection “isBasedOn” cannot be document object itself. Similar operation rejectAll() could be used to except a collection of objects from related collection. On the contrary, operations include() and includeAll() could be used to preserve an object or object collection respectively to be in related collection.

Consider pre-existing constraint, requiring correspondent object existence before anchor object appearance: “Every outgoing document must be linked with some document”. The OCL expression for this example could be conveyed in the following way:

\[
\text{context Document} \\
\text{inv: self.kind=DocumentKind::outgoing} \\
\text{implies self.isLinkedWith->notEmpty()}
\]

The expression above states that the fact that “document is kind of outgoing” implies the collection of objects “isLinkedWith” to be not empty. It follows thence that while the document is kind of “outgoing” it must be linked with other document.

### 2.4 Expressing type verifiers

Type verifiers control the creation of multiple instances in various object classes [3]. Type verifiers control occurrence of objects in object classes and may be one of four types: mutual (requiring that correspondent objects exist simultaneously), mutually exclusive (requiring that no more than one correspondent object exist simultaneously), mutually dependent (requiring that either one instance of every correspondent
object class exists, or that no instances of any correspondent object class exist) and mutually prohibited (requiring that at least one of the correspondent object classes has no instances).

For this type of rule comparison of collections may be employed. Combination of operations on collection with logical operators (OCL support and, or, xor operators) could be used to express most of this type of rule. Consider following rule “Document can be created only by one of following: a user, a users group, a contact or a contacts group”. This rule could be referred to mutually exclusive subtype of type verifier. Depending on example model, the document is created by some author which can be a user, a user group, a contact or a contact group. The following OCL expression mutually excludes candidate subjects to author:

\[
\text{context Document}
\text{inv:}
\text{self.createdBy.user->notEmpty()} \text{ and } (\text{self.createdBy.userGroup-> isEmpty()} \text{ and self.createdBy.contact-> isEmpty } ) \text{ and self.createdBy.contactGroup-> isEmpty() } )
\text{ or}
\text{self.createdBy.userGroup->notEmpty()} \text{ and (self.createdBy.user-> isEmpty () and self.createdBy.contact-> isEmpty () and self.createdBy.contactGroup-> isEmpty() )}
\text{ or}
\text{self.createdBy.contact->notEmpty()} \text{ and (self.createdBy.user-> isEmpty () and self.createdBy.userGroup -> isEmp...}

The invariant above disjunctively joins four rules stating that one collection must not be empty while others must. In the same manner could be expressed other types of type verifiers. Consider mutually prohibiting rule: “Internal document cannot be received by or send to any subject(s)”. Corresponding OCL expression could be modelled as follows:

\[
\text{context Document}
\text{inv: self.kind = DocumentKind::internal implies self.sentTo -> isEmpty() and self.receivedBy -> isEmp...}
\]

The invariant above involves implication operator and comparison of the collections. It states that document of kind “internal” implies associated collections of objects sentTo and receivedBy to be empty.

2.5 Expressing sequence verifiers

According to [3], sequence verifiers control changes in object state. If object may be in multiple states then these rules determine the sequence in which instances of that object class may assume those states.
Sequence verifier may be of one of the following types: initialising (requiring some state on object initialization), forward (requiring transition to a higher state of object), progressive and retrogressive (requiring transition to a next higher or lower state of object accordingly), re-initializing (defining that when object moves to a lower state it should be moved to initial state first) and cyclical (defining that object can be moved to a lower state before it moves to highest state and vice versa).

This type of rules cannot be fully defined in OCL, because OCL is a declarative language. The best way to express changes of object states in UML is state machine diagrams [9]. UML state machine diagrams describe the behaviour of a class over time of the states and transitions of a single object progressing through its lifetime. In UML state machine diagrams OCL expressions may be used in a number of ways [4], but commonly used are guards on states transitions and restrictions on states. The guard is condition on transition in a state machine diagram that must be met to change a state of object. Usually, restrictions on states are restrictions on values of links and attributes when an object is in a certain state. Using UML state machines to define a sequence of object states and OCL to express restrictions on model elements is possible to express most of the rules of type of sequence verifiers.

Referring to considering example, document lifecycle could be explained as state machine diagram as in Figure 8:

![Figure 8. State machines diagram of document states](image)

The process is as follows: A new document saved in system is preparing for release. After acceptance it may be suggested for release. Afterwards suggestion for release document may be approved or rejected. If document is approved, then it could be released; otherwise, if document is rejected, the new version of document must be created referring to the cause of rejection. As requires sequence verifiers, sequence of object states is defined (by state machine diagram) in model. Mentioned-above guard on transition is described on transition from the state draft to the state final, which is entry point to release process by UML notation meaning. The restriction is enclosed in brackets and denotes that transition is possible only if document is prepared (by checking Boolean type attribute isPrepared value). Restriction on state is described on releaseCandidate state denoting that document can be in this state only if it is suggested (by checking Boolean type attribute isSuggested value).

Considering to sequence verifier subtypes, control of object states could be preserved by operations on any OCL instance `oclInState()` or `oclInState (str:StateName)`. Suppose following rule: “Every newly created internal document is draft until it is accepted that document is prepared for release”. Firstly, the rule denotes that initial state of document “draft”. Secondly, after acceptance, document must be prepared and in “final” state. Corresponding OCL expression would be:

```ocl
context Document::accept()
pre: self.oclInState(draft)
post: self.isPrepared=true and self.oclInState(final)
```

OCL expression above restricts operation `accept()` denoting that before execution of the operation document must be in draft state, and after it must be prepared and in final state. Due to the fact that objects state changes due to execution of some related operations, control of execution of such operations may be guarantee via pre and post conditions on object operations. The combination of pre and post conditions with the operation `oclInState()` may be applied on expressing the control of sequence of object states in state machines model.
2.6 Expressing position selectors

Position selectors pertain to a value, either in a value sequence or a chronological sequence [3]. This type of rules is of two types: positioned, lowest and highest and chronological, oldest and newest. Suppose the rule: “Document version must increase by one on set of version”. Respective OCL expression could be modelled as follows:

```ocl
context Document::setVersion(version: Integer)
pre: --none
post: version=@pre + 1
```

Expression above denotes that after execution of operation `setVersion()` document attribute “version” must increase by one. OCL support keyword @pre, which refers to an attribute value at the start of operation. Employment of this keyword may ensure positioned sequence of objects in the collection; besides, OCL support loop operations on collections. Depending on the nature of BR, different types of loop operations may be applied. For example, the rule mentioned above could be expressed using iterator `forAll()`:

```ocl
context Document
inv: self.allInstances->forAll(doc1,doc2 | doc1 <> doc2 implies doc1.version = doc2.version + 1)
```

Iterator `forAll()` returns true if for all elements off the collection, used test condition is true. In this case the operator `forAll()` is used to denote that for every different pair of elements in the collection of the document objects, object (doc1) attribute’s “version” value must be greater by one than compared one (doc2). In the analogous fashion, expression of positioned type of position selectors could be applied to other business rules.

However, chronological sequence rule requires additional constructs to be involved in model, because OCL does not support date or time data types. Therefore, there should be involved some utility elements to support chronological sequence of elements of the collection in model.

2.7 Expressing functional evaluators

Functional evaluators take care on sequence in which instances of object class are defined [3]. There are following types of functional evaluators: unique (requiring unique sequence of values), ascending and descending (requiring sorting of values in sequence), non-renewable (requiring that any given value of the correspondent object, if used more than once, may be used only in strictly successive instances of the anchor object), patterned (requiring that successive instances of the anchor object class be assigned in a specified sequence, or tests for that condition).

As it was discussed in previous subsections, control or modification of sequence of elements could be preserved using iterators. For example unique values evaluator could be the following rule: “Document registration number must be unique”. Respective OCL expressions could be:

```ocl
context Document
inv: self.allInstances->isUnique( doc | doc.registrationNumber )
```

, or

```ocl
context Document
inv: self.allInstances->forAll( doc1, doc2 | doc1 <> doc2 implies doc1.registrationNumber <> doc2.registrationNumber)
```

Expressions above are identical, because the first one uses operator `isUnique()`, which tests if every document has a unique registration number, and the second one tests whether for every pair of different document objects registration number is not the same.

Sorting of collection elements could be modelled using `sortedBy()` which simply sort collection’s element in ascending mode. For example, the rule „Document registration number must be in ascending order within a file” could be expressed by defining attribute `sortedDocuments` of collection type in context of file:

```ocl
context File
inv: sortedDocuments->sortedBy(doc -> doc.registrationNumber)
```

Figure 9. Class diagram of rule type of functional evaluator
Expression above collects sorted elements of documents collection into sequence. The sequence as well as other type of OCL collections defines a sequence and ordering of elements of collection. Therefore if it is requiring that collection elements should be served out in some pattern operations asSequence(), asBag(), asSet(), asOrderedSet() could be applied [11]. Combining mentioned operators with iterators on collections could produce sequences of elements arranged in predefined way thus providing possibility to express complex functional evaluator rules.

2.8 Expressing comparative evaluators and calculators

Comparative evaluators describe comparisons between pairs of instances of object classes. The comparisons may be 'equal to', 'not-equal-to', 'greater-than', 'greater-than-or-equal-to', 'less-than', or 'less-than-or-equal-to', and so forth [3]. OCL supports different logical operators for this purpose and some of them were discussed above. It should be noticed that OCL is strongly typed language, therefore compared objects or values should be carefully chosen in comparison of object values.

Calculators involve any standard computation, e.g. sum, subtract, max, min, med, etc. Calculations in OCL are possible between two data types: integer and real. Calculators may be expressed using standard OCL calculating operations (summary, subtraction, division, modulus, etc.) but as well as in the case of comparative evaluators conformance of types is required.

2.9 Expressing update controllers

Update controllers prescribe whether updates to a database may occur and may be of the following types: frozen (requiring existence of anchor object to make some operation to correspondent object), frozen to users (the same as above, but restricted to specific list of users), enabled (existence of every instance of anchor object enables operations on correspondent object) and enabled with reversal (when anchor object is deleted the state of correspondent object is reversed).

OCL is the constraint language therefore every expression in some way may constraint an operation to a database. In object-oriented systems object attributes values are set or get by some defined operation. Therefore if it is required to restrict some changes of value then pre and post-conditions on operations could be applied. For example, restriction on change of value may be usage of post condition on operation denoting that value must be the same as at start of operation by introducing @pre keyword as it was shown in previous example.

Certainly, if it is required to relate constraints on operations with some users, e.g. restrict execution of operation or change of value to specific users, additional constructs should be introduced into the model to check whether operation is executed by related user or not.

Consider following rule “Only assistant can register internal documents” requiring an assistant role to execute the operation register(). One of possible expressions of this rule may be introduction of new operation into document class. The operation, e.g. checkRole(event: String, role: Role) could be of Boolean return type checking whether some event is made up by some role.

Additionally, OCL support hasSend('^') operator, which is used in post condition of operation to specify that communication has taken place. Consider rule stating that “If file is deleted, then every involved in file document must be deleted as well”. Appropriate OCL expression may be constructed using hasSend() operator as follows:

\[
\text{context File}::\text{delete()}
\]
\[
\text{post: consistOf^delete()}
\]

Expression above states that after execution of operation delete() of File object, operation delete() of Document related to file Document object must be executed. Hence, with additional tests, the control of updates...
may be vouched by testing the results of related operations (post-conditions) and stimulating or preventing execution of another operation.

2.10 Expressing timing controllers

Timing controllers prescribe tests for the length of time that instances of correspondent objects have (or have not) existed [3]. OCL does not support any constructs to express time-based constraints to control objects states in time. Therefore, additional elements (such as time utility) should be introduced to model or extension to OCL could be provided as suggested in [6] [12].

3 Conclusions

In this paper, research on possibilities to express different types of action assertion business rules with OCL was made. It was established that OCL is expressible enough to capture most of action assertion type rules. Operations on collections of objects may be used to express instance verifiers. Operations on collections in combination with logical operators or calculative operations could be used to express type verifiers as well as functional evaluators. Pre and post conditions may be used when modelling sequence verifiers or position selectors. The combination of pre and post conditions with the operation for test of object state may be applied on expressing the control of sequence of object states in state machines model. Reluctantly, standard OCL cannot express some types of business rules, therefore suspending of model with additional constructs is required or extension of OCL may be proposed.

4 References

MANAGEMENT OF BUSINESS RULES USING DECISION TABLES*

Kestutis Normantas, Olegas Vasilecas, Sergejus Sosunovas

Vilnius Gediminas Technical University, Information Systems Research Laboratory, Saulėtekio al. 11, LT20221, Vilnius, Lithuania, {kestutis.normantas,olegas,sergejus}@isl.vgtu.lt

Abstract UML has established itself as the leading OO analysis and design methodology. This paper presents a business rules engineering approach to define decision tables in consistency with Unified Modeling Language (UML) model. Supporting tools are under the development on top of a commercial CASE tool for UML.

Keywords: Business rules, decision tables, UML, OCL

1 Introduction

Business rules are evidently important for organizations as they describe how they are doing business. Their value has also been recognized within the information system (IS) domain, mostly because of their ability to make applications flexible and amenable to change [1]. During the last years together with increasing importance of research in semantics the interest in rule based software systems and their development is renewed and increasing. This results in a numerous rule engines available today on the market.

Decision table is one of the oldest techniques to enter business rules in the rule engine [2]. Official decision table components and terminology were standardized by the cooperative efforts of a number of organizations, including the Conference on Data Systems Languages (CODASYL) in 1962. The basic format of decision tables remains largely unchanged today [3]. At the moment this method is successfully used in the leading business rules product (e.g. Corticon BRM Platform, Fair Isaac Blaze Advisor, ILOG JRules, Resolution EBS iR Platform etc.) [4]. Decision tables are a form of structured lookup capability that provides the ability to construct and update conditions and actions for rules as a cross-indexed chart or table.

It is important to notice that business rule engine cannot be used alone; more often it is integrated into a bigger information system. The recent research on the development of business rules based systems reveals that the half of the business rules systems contains not more than 120 rules and takes 9 man months to develop. At the same time at the average the rule base is part of a much larger software system that takes almost 60 person months to develop [5]. Therefore the methods to develop and present business rules systems should be seamlessly integrated with the methods used to develop general purpose software systems.

UML is a popular way to model software system. In order to model information system using UML it is necessary to introduce a way business rules in the form of the decision tables can be integrated with UML model. In this paper we advocate an approach of using metamodel based integration of decision tables and UML. The necessity to use decision tables with UML is mentioned in [6].

This paper starts by giving a brief overview of related work. Then in Section 2, it gives an overview of proposed method for business rules specification and implementation in modeling process. Next in Section 3, it illustrates the approach by applying it to a simple problem. Finally in Section 4, the paper discusses the results of the research presented here and makes the conclusions.

2 Related work

Business rules are important in the modeling of business system: they help to define the business terms and facts (structural assertions) as well as the constraints underlying the business behavior (action assertions). According to [7],[9] business rules represent core business policies that capture the nature of an enterprise’s business model and define the conditions that must be met in order to move to the next stage of the process. Business rules are represented as compact (declarative) statements about an aspect of the business that can be expressed within an application in unambiguous terms that can be directly related to the business and its collaborators and as such they determine the route of action to be followed [8].

Business rules can be integrated with enterprise applications so that they can be used for business decision making, using ordinary business data. Business rules, in general, automate and facilitate business processes. They allow business analysts and even users to create, understand and maintain the rules and policies of the business and associate them with relevant business processes. They are usually grouped into independent but chainable rule-sets and perform inferences within and over such rule-sets [9].

* The work is supported by Lithuanian State Science and Studies Foundation according to High Technology Development Program Project “Business Rules Solutions for Information Systems Development (VeTIS)” Reg. No. B-07042
Since its introduction, UML has quickly been adopted as the standard Object-Oriented (OO) modeling language for modeling software systems [16]. The Object Constraint Language (OCL) [10][11][12] was introduced to supplement UML to capture more details of modeling domain. Nowadays, UML/OCL is supposed to be used for specification of business systems as well and it seems that popularity and number of tool supporting business related UML notation increase. CASE tools, such as MagicDraw UML, support easy construction of models which capture business processes and rationale. Once embedded in the model, however, this information becomes somewhat inaccessible. In order to be able to extract the information from the model one generally needs a working knowledge of the UML/OCL and some knowledge of the operation of the CASE tool. Thus UML lacks both a reference implementation and a human-readable semantic account to provide an operational semantics [6].

We suppose that decision table technique could partially fill this gap. Because of this technique is easily integrated with other techniques, binding with UML/OCL would not produce any problems in the modeling process: the UML provides three mechanisms for extending the language's syntax and semantics: stereotypes (which represent new modeling elements), tagged values (which represent new modeling attributes), and constraints (which represent new modeling semantics) [16]. On the contrary, it would supplement OO modeling technique with understandable for non-technicians form for specification of subset of business rules. As a result of decision table constructing, different artifacts may be produced, such as OCL expressions or activity diagrams. Of course, reverse binding is possible as well and will be provided in further research. Unfortunately, limited subset of business rules can be modeled with decision table technique. This includes conditional constraints, derivations and process rules [15].

Binding of decision table technique with Entity Relationship (ER) seems to be successfully realized in research made in [13]. In this research was used metamodel based transformation approach to produce ER model from DTs. It was shown that decision tables are applicable for business rule specification stored in XML using decision table metamodel. Related approach could be adopted to recent research as well.

A decision table is a business rule in form of table, which consists of a set of conditions and a set of actions or sometimes called conclusions. Conditions usually are represented as rows and actions as intersection points of the conditional cases in the table. Decision tables are best suited for business rules that have multiple conditions. Augmenting table with another condition is done by simply adding another row or column. In a decision table, the action is decided by more than one condition, and more than one action can be associated with each set of conditions. If the conditions are met, then the corresponding action or actions are performed.

Though originally used as a technique to support programming, decision tables have proven a useful aid in modeling complex decision situations of various sorts. In the field of knowledge-based systems research, there has been a renewed interest in decision tables over the past years. Here, decision tables have been studied or applied in the following contexts: verification and validation of knowledge-based systems, efficient execution of knowledge-based systems, knowledge base maintenance, knowledge acquisition, knowledge discovery, several application domains such as medicine, law, etc [14].

Because of so large interest in decision table technique, different kinds of them originate. The main difference is depending on whether all columns are mutually exclusive or not. If so, then each possible combination of conditions matches exactly one column. Otherwise, some combinations of conditions match more than one column, which may lead to ambiguity or inconsistency. In such case sequence of table conditions hits (selection) is very important: the first hit (when scanning the table from left to right) will determine which set of actions has to be executed, thus preventing contradictions. In this paper, we stand with mutually exclusive arrangement of columns as it will be shown in the next subsections.

3 Decision table based approach for business rules representation and implementation in OO modeling process

3.1 Decision table construction process

The main idea for binding of decision table technique with UML/OCL is to invoke business people to participate in the software development process in the earliest stages and to talk to them in understandable to them form. Therefore, participation of business related people (e.g. stakeholder) in construction of decision table could produce precise understanding of particular business rules.
Figure 1 represents decision table construction process. The first step in the construction process is identification of condition labels. In the simplest case condition labels may be either attributes names, or operations (which return some result) names. The second step is to identify conclusions (or actions). Conclusions as the conditions may either attributes names, or operations names. The third step is identification of rules. In this step possible alternatives of condition labels are in consideration. Identification involves filling of possible values for condition labels and introduction of new alternative rules in the model. After, if it is required, then reordering of the condition labels is made. In the next step conclusions are filled according to formed rules. After it, validation of decision table with stakeholders could be applied to check whether the constructed table is appropriate to business rule. In the case if it is not so, additional check of decision table construction should be provided.

3.2 Business rules specification and artifacts generation

To illustrate our approach consider a simple problem. Calculation of delivery price depends on the price of the order, preferred delivery time and on the customer status in business accounting system. Delivery time may be a month, a week or a day. Customer status may gold, silver or bronze. Related class diagram is presented Figure 2.

Possible delivery prices according to mentioned above parameters are presented in decision table (Table 1). Decision table’s abstract sphere consists of condition labels and appropriate actions. For this example, it is composed of only attributes of classes for the simplicity of the example; additionally, it may be composed of operations of classes or OCL expressions returning values to be assigned to the concrete sphere’s rules.
Table 1. Decision table for calculation of delivery price

<table>
<thead>
<tr>
<th>Abstract sphere</th>
<th>Concrete sphere</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order price</td>
<td>Order price</td>
</tr>
<tr>
<td>Delivery time</td>
<td>day</td>
</tr>
<tr>
<td></td>
<td>week</td>
</tr>
<tr>
<td></td>
<td>month</td>
</tr>
<tr>
<td>Customer status</td>
<td>G</td>
</tr>
<tr>
<td>Actions</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Rules</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

* (G-gold,S-silver,B-bronze)

The result of decision table depends on the related model elements where decision table is applied. In current example it is applied to deliveryPrice attribute of the class Order. Thus result could be invariant on order class or initialization expression of deliveryPrice attribute. OCL invariant generated from decision table would be:

```ocl
class Order {
  inv deliveryPriceConstraint:
    if self.price < 1000 then
      if self.deliveryTime=DeliveryTime::day then
        if self.customer.status=Status::gold then
          self.deliveryPrice = 10
        else
          <...>
        endif
      else
        <...>
      endif
    else
      <...>
    endif
}
```

As we can see from the above example, description of simply problem may become complicated enough when it is expressed by if-then-else statements. Modification of condition value would lead to modification of OCL expression in general. However a table with many different parameters quickly becomes complex what diminishes its usefulness as a simplification concept. Therefore additional algorithms should be invoked to optimize it as well as friendly graphical user interface for filling of decision table form should be provided.

If table is applied to an operation of a class, then result may be a post-condition or a body of an operation. Construction of OCL expression would be very similar to expression above. Result of an operation may be activity diagram as well, as shown in Figure 3.

![Figure 3. Fragment of resulted activity diagram from decision table](image-url)
actions setDeliveryPrice with different input pins which are evaluated depending on activity flows. Regarding to OCL usage with UML, guards on activity flows as well as input pin could expressed as OCL expressions.

3.3 Mapping of decision table metamodel to UML metamodel

We have used simplified MOF metamodel for decision table presented in [15] and MOF metamodel for UML to show how decision table could be used to specify business rules for a class diagram. Additional mapping of decision table metamodel with OCL and UML for activity diagrams will be provided in further research.

![Figure 4. Mapping of decision table metamodel [15] with UML](image)

Figure 4 presents suggested mapping of decision table metamodel elements to UML metamodel elements. The composition of concrete sphere’s rule set consists of an ordered composition of condition labels. These condition labels group the conditions of the rule set in sets of exhaustive and mutually exclusive conditions. Each condition is a logical formula that refers to the domain atoms and variables of the rule set. The actual rules of the rule set are an ordered conjunction of conditions, such that a rule contains at most one condition of each condition label. Notice that not every conjunction of conditions is necessarily meaningful. In other words, it might be the case that a specific condition is only meaningful in combination with other specific conditions. To express this dependency, a decision table can make use of so called condition dependency rules [15]. UML metamodel, used in CASE tool MagicDraw UML, fragment represents the structure of the class. Class consists of operations and properties. Operation consists of parameters and is extended by setter and getter stereotypes. Property (an attribute) is extended by propertyGroup, hasGroupName and suggestedValues stereotypes. As it was mentioned before, conclusion and condition label could be mapped to operation and property metaclasses. For the implementation we use those metaclasses instances to construct decision table.

3.4 Experimental evaluation of the approach

Currently, proposed approach is implementing as the plug-in for the CASE tool MagicDraw UML (Figure 5). It is in the earliest stage of the development, therefore only generation of invariants on attribute values has been implemented.
In the current implementation, decision table may be called either from common actions toolbar, or from class context menu. Called from context menu, decision table has already assigned context class for generation of invariant. In other case, context class should be assigned configuration settings. Decision table conditions as well as conclusion values may be selected/removed from model elements selection window. Values for conditions or conclusions are inserted manually. At the moment, refinement of graphical user interface for decision table filling form as well as development of plug-in to result more artifacts is in progress.

3.5 Discussion

Proposed approach showed that decision table may simplify modeling a subset of business rules with UML/OCL. Of course, decision table is one of many techniques which could be used with UML/OCL to represent business rules to business related people in familiar to them form. We have shown that decision table may be used as graphical user interface for modeling particular subset of business rules to provide OCL expressions. Resulting artifacts of decision table is an open question and topic for the further research. Implementation details of OCL are not in scope of this paper, because there was presented the main idea; therefore, no related algorithms were discussed here. Algorithms for conversion of decision tables to OCL invariants and to activity diagrams will be published in the future.

4 Conclusions

In this paper we have set out the usage of decision table technique with standard OO modeling method UML to provide familiar to business people form for specification of particular subset of business rules. We have shown that decision tables usually used as business rule visualization technique are applicable for business rule specification with OCL.

The proposed approach allows expressing business rules in user-friendly manner as well as supplementing related UML model by adding additional elements when construction of decision table is being processed. All the limitations of the decision tables belong to the proposed approach too and should be solved using other engineering methods.

References


A PERSONA METHOD ADAPTATION FOR REQUIREMENTS ENGINEERING IN BUSINESS INTELLIGENCE REPORTING

Maikel Peréz Gort¹, Peter Thanisch², Jyrki Nummenmaa³, Zheying Zhang²

¹CUJAE(Jose Antonio Echeverria Higher Technical University, CEIS (Centro de Estudio de Ingeniería de Sistemas), Havana, Cuba, mlazaro@ceis.cujae.edu.cu
²University of Tampere, Department of Computer Sciences,33014 University of Tampere, Tampere, Finland, firstname.lastname@cs.uta.fi

Abstract. The persona method has been successful in requirements engineering for systems in which a user interacts with the system as an individual, effectively oblivious to other users. We describe our adaptation of the Persona method which facilitates requirements engineering for collaborative systems. We have applied this adaptation in a requirements engineering study of business intelligence reporting platforms, where usage can take the form of a complex sequence of collaborations between a heterogeneous collection of team members. We found that the main shortcoming of the existing persona method is that it pigeon-holes users into static roles. For our purposes, however, a method of analysis must (a) account for users' aspirations and (b) take a dynamic view of usage within an organization over time and (c) give insight into the tensions in the collaboration between personas.

Keywords: requirements engineering, usability, business reporting, persona

1 Introduction

Business intelligence (“BI”) reporting is the most important area of BI, with respect to both expenditure and perceived value by end users. Although there is no standard definition of BI reporting, we can characterise reports and reporting applications as follows [18,21].

“A report extracts data from back-end data sources (e.g. databases), uses business logic to convert the extracted data into useful information and presents the information in a meaningful way to a business decision-taker.” [21]

Our characterisation of a report deliberately avoids stating anything about the delivery format (paper or electronic), the usage (interactive or passive), or the production process (program coding or graphical design tool). In everyday language, the term ‘report’ refers simply to the output that the user sees. However, in practically all technical literature on reporting technology, the term ‘report’ can also refer to the design of a report and we shall follow that practice in the present paper. Even restricting our attention to a canned report, the output can contain fairly complex interactions, such as drill-down and sorting.

Almost every medium to large organization uses at least one BI reporting platform. Although the major reporting platforms are mature and stable products, they are still evolving in response to both technological change and demands from an increasingly-sophisticated user community. The usability of reporting platforms is an interesting topic because the platform architecture embodies a hierarchy of design environments, each optimized for a particular set of users. At each intermediate stage in this hierarchy, the human designer uses the design artifact produced by a colleague at the previous design stage in conjunction with a customized design environment provided by the reporting platform (see Figure 1). Typically, these intermediate design artifacts are intended to be re-used, possibly many times, in the production of different reports by different users.

Over the years, in parallel with the development of this platform architecture, a complex division of labor has evolved in those enterprises that make an extensive use of reporting. In the authors’ experience, most organizations are dissatisfied with the time and resources needed to produce a new BI report. We believe that many of the issues associated with the process of creating and maintaining business reports are exacerbated by tensions that arise from the mismatch between the notional division of labor implicit in the platform’s architecture and enterprise’s actual division of labor. The goal of our research is to identify ways of resolving those tensions by means of a requirements engineering exercise.

Our chosen requirements engineering technique the persona method [8], is recognized as facilitating designs that make systems-to-be-built more “user-centric”. We give a fuller account of the method below, but briefly, the method involves identifying a set of clusters of user characteristics and from these, concocting a small set of personas deemed to be representative of the significant aspects of the user community. Taken collectively, the set of personas thus identified is called the cast [2]. The persona method has been developed for
Systems in which the users interact with the system individually, rather than collaboratively. There may be other multiple concurrent users, but each individual user is largely oblivious to the others as they interact with the system. For our investigation of business intelligence reporting platforms, we need a version of the persona method that takes into account complex collaborative patterns between the personas in the cast.

This paper describes our adaptation of the persona method for use in analyzing requirements for BI reporting platforms. We have piloted this adaptation in a requirements engineering exercise for a popular Cuban information and reporting system [15,16]. However, the present paper is work-in-progress rather than a description of a finished product. In our experience of applying the persona method to BI reporting platforms, the main shortcoming of the existing method is that it pigeon-holes users into static roles. For our purposes, however, a method of analysis must (a) account for users' aspirations and (b) take a dynamic view of usage within an organization over time and (c) give insight into the tensions between the team members individual goals and the collective goal of the team. Our adaptation goes some way to addressing each of these issues.

The content of the rest of this paper is as follows. Section 2 describes reporting platforms in general then goes on to describe the GREHU reporting platform, the object of our study. Section 3 contains a description of relevant aspects of the persona approach. In Section 4, we present the results obtained from applying the persona approach to the usage of the reporting platform. We conclude with some remarks in Section 5.

Unfortunately, there is no industry-standard terminology for describing reporting platforms. Rather than invent our own terminology, we adopted the terms used by one of the platform vendors. This choice of terminology does not reflect a platform preference on the part of the authors.

2 Reporting Platforms

2.1 Platforms in General

Although there is a large number of reporting solutions on the market, many of them are specific to a particular niche market and some are designed to be used by programmers. In the present paper, we focus on reporting platforms, i.e. general-purpose reporting solutions that include explicit graphical user interfaces for tasks such as data definition, layout design, design of internal properties (such as interaction features) and the publication and administration of reporting [1, 11, 13, 23].

In practice, there is often a complex division of labor associated with the use of a reporting platform. The terminology varies between vendors and details vary from one customer site to another, but Table 1 gives a representative set of job titles, based on those used by a reporting platform vendor [23].

<table>
<thead>
<tr>
<th>Job title</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report consumers</td>
<td>They actually use the information in the report. They know about the business, but they do not necessarily know about the data or the reporting platform.</td>
</tr>
<tr>
<td>Power users</td>
<td>They have some knowledge of the business and also sufficient knowledge about the reporting platform and the data source view to facilitate a degree of self-service.</td>
</tr>
<tr>
<td>Report Authors</td>
<td>They have a limited knowledge of the business and an in-depth knowledge of the design facilities of the reporting platform.</td>
</tr>
<tr>
<td>Modeller</td>
<td>They know the reporting platform architecture and have a high-level understanding of the overall information needs of the application.</td>
</tr>
<tr>
<td>Data specialists</td>
<td>They have a very limited knowledge of the business. They know how to extract data information from the data sources (XML, OLAP, relational, etc.). They may not know much about the reporting platform.</td>
</tr>
</tbody>
</table>

Figure 1, below, depicts the set of artifacts and interfaces involved in the end-to-end production of a report. At each intermediate stage, the user uses a design artifact produced by a colleague at an earlier stage. Apart from the end user, each individual uses the design facilities provided by the reporting platform vendor for his or her interface to produce a design artifact for a colleague further along the design chain. The rationale for this complex division of labor is that the designers at the early stages are the people who know a lot about the

---

1 Our list of reference to reporting platforms is intended to be representative rather than exhaustive.
data and not so much about the business, whereas the designers at the later stages know a lot about the business and not so much about the data.

Figure 1. Artefacts and interfaces in a reporting platform
Figure 1 depicts a special case of the report production process. In general, a new request for a report might be satisfiable by re-using or adapting an existing data source view or model. Good designers successfully anticipate such future demands when producing design artefacts.

The simplest example of the reporting cycle is illustrated by the sequence of events depicted as Steps 1 to 4 in Figure 2. Ideally, this might complete the report. However, typically there is an iterative reporting cycle [21] which additionally includes the following steps. We have depicted the reporting cycle in Figure 2.

![Figure 2. A simple reporting cycle, involving just two actors](image)

Figure 2 depicts the simplest reporting cycle. A more typical reporting cycle involves more of the actors in Figure 1. Typically, each additional actor involved in a reporting cycle has a considerable impact on the elapsed time to complete the cycle. This is due to e.g. various report testing procedures as well as just waiting for the required assistance to complete the required changes to a report. In many organizations, people with report design skills are very scarce and there is a long queue of report requests.

Some reporting platform vendors claim to use an approach akin to the persona method [6], but the outcome has been a simplistic, top-down mapping of personas to roles. The sweeping assumptions do not just affect the platform architecture and user interfaces. The vendors incorporate these assumptions in other areas such as training, documentation, licensing and even in automatic workload generation for testing. To some extent, vendors can force reality to reflect their assumptions, in the sense that enterprises using BI reporting platforms feel constrained to organize their reporting activity to match the platform vendor’s preconceived notions. We believe this is a tension-point and that is why we decided to look again at the persona method.

2.2 The GREHU system

For almost twenty years, the Center of Engineering Information System (CEIS) of the University Center “José Antonio Echeverría” (CUJAE), Havana, Cuba, has been developing and maintaining an information system to improve management of human resources. The system, called GREHU (Gestion para Recursos Humanos) has been designed according to the Cuban regulations and practices, and it has been maintained to keep up with legal changes, as well as changes caused by the dynamic nature of the application area.

Nowadays, GREHU is considered a highly complex system. It has grown over time, including now several different sub-areas of human resource management. At present, GREHU is composed of 16 subsystems and is used by more than 250 Cuban enterprises. The overall structure of the GREHU system is shown in Figure 3. Some general and feature-specific information about GREHU can be found in [15,16,19,22].

GREHU must provide a highly dynamic management of reporting because of continual changes in reporting format required by users and by new legislation that impacts on the requirements for reporting models. Demands for new reports can require at least some level of system-wide reprogramming, with consequent delays to release schedules. This inconveniences clients who are forced to use incomplete information or obsolete report forms, possibly leading to incorrect conclusions.

The GREHU development team provides day-to-day support to clients, distracting them from development. Whereas legislative changes can affect all the clients, some individual clients may have particular requirements that need separate attention.
The main causes of delay in satisfying client requirements for report changes and the creation of new reporting designs are:

- The maintenance tasks and capacitiation of users for the old system demands too much time and makes it difficult to maintain the momentum of research and development activities for new solutions and software products for the GREHU develop team.
- The other maintenance tasks, unrelated to reporting, may demand prior attention, for example, salary processing.
- When different clients demand different reporting changes in the GREHU system, there is inevitably a queue of requests.

Attempts to tackle the main problems related to reporting that GREHU presents include the development of a new subsystem called Reporting Generator. However, sometimes the solution to these problems must be sought outside of the reporting subsystem. We consider that the reporting needs must be considered from the beginning of system design.

A key design goal of GREHU’s reporting subsystem is simplify the process of making changes to report designs. Although this increases user satisfaction, the inclusion of this subsystem may lead to new issues emerging. Furthermore, those issues may be harder to resolve than keeping up with the pace of change demands of reporting designs. These new problems can be more profound and they may even prohibit adjustment of the system for some organisations.

In GREHU report management there are different user roles that make the processes more efficient and make the system more applicable. This is, of course, beneficial as different users differing capabilities and knowledge to work with reporting. However, a more detailed study of user roles and capabilities in reporting organisations is worthwhile because it can lead to the development of software to provide better support for the individual capabilities of different users, thereby allowing us to make better use of reporting.

### 3 The Persona Approach

The persona method has been employed in user-centered design in order to define the requirements for a system, focusing especially on the suitability of user interfaces. Many studies [2,3,4,9,10,20] demonstrate the utility of personas in user interface design. In our attempt to model how reporting organizations exploit their reporting platforms (and, indeed, how they should exploit them) we use Aoyama’s Persona-Scenario-Goal (PSG) method [2,3]. We briefly define those three terms.

**Personas** are fictional characters constructed to be representatives of a group of users. Each persona encapsulates a distinct set of usage patterns and behavior patterns in relation to the system-to-be. Additionally,
the persona is ascribed a set of human attributes like gender, age, race, ethnic, family or cohabitation arrangement, socio-economic background, work, or home environment [20] so that it takes on the character of a real representative of a group of users. This makes the persona feel more concrete and less “elastic” to product engineers than a description of a user role [9]. In general, each product has one primary persona to represent the primary target for the design of an interface, but it is possible for some products, such as enterprise products, to have multiple distinct interfaces targeting at several primary personas [9]. Besides, there are personas representing the groups which are only indirectly related to a product or its use (a ‘served’ persona) and even those that are hostile to the product (a ‘negative’ persona) [9].

Scenarios are contexts in which a persona makes use of the system-to-be-built. For some applications, one or several scenarios can be represented by a use case. For our purposes, where usage of the system involves creativity and design, a use case may comprise different sets of scenarios in distinct contexts. So scenarios are more appropriate to associate with personas and their goal than use cases. A scenario is the context of the activities and events by which a personas achieves a goal. It is commonly used in a method of design problem solving by concretization [7] making use of a specific story to both explain the problems and construct design solutions. In general, goals serve as a filter for tasks and as guides for, e.g., structuring the display of information and controls during the iterative process of constructing the scenarios [9] and eliciting requirements.

Goals are used to evaluate requirements and analyze conflicting requirements [3]. A goal is an explicit statement of the aim or purpose [24] that exists in every persona activity. The goal represents the persona’s individual needs and expectations. According to Cooper and Reimann [9], goals and not features are the key to product success. In practice, developers tend to focus on features rather than goals, since features relate more obviously to their technology and methodology [12,14,17].

The PSG method improves our understanding of the problem domain by taking into account heterogeneous groups of users, their needs and behaviors in the system development process. It helps to acquire requirements for mass-users products, especially to evolve the existing prototype according to the goals. The scenarios will be analyzed and evaluated from the perspective of the primary persona and in the light of the identified goals. When an issue is found in a given scenario-goal pair, a note will be recorded. New requirements are elicited and refined based on the issues. In the process, goals are used as the main criteria to resolve conflicting and inconsistent requirements. In general, the PSG method is useful in refining requirements for an existing product or a prototype, as a preliminary set of scenarios and goals shall be defined beforehand. Thus it is applicable in designing a reporting platform based on existing reporting systems.

In this paper, our main focus is on using the PSG method for constructing personas for requirements elicitation. This is achieved by the following procedure [2,3].

1. **Identify and Select Sample Users:** A set of sample users is chosen on the basis that they are representative of a cross-section of the overall user group, that is, they possess the characteristics that are important from the point of view of the system being developed or analyzed. The selected sample users are interviewed.

2. **Identify User Groups:** Based on the interviews, the people from the sample are divided into groups. Each group has a similar patterns regarding the use of, or need for the product. Each of these user groups represents a type of user.

3. **Identify Features’ Value:** A set of proposed features of the product is provided for user groups’ evaluation. These features can be obtained from the high-level objectives of the organization or from customers who request the product. Each group rates each of the proposed features. The list of features is adjusted based on the results. For instance, if none of the groups values a certain feature, it can be removed.

4. **Identify Primary Persona:** The primary Persona is the one that uses the most features.

After identifying primary personas, the PSG method leads the requirements engineer to describe features in scenarios and elicit requirements for individual features from the perspective of the primary personas.

Even when the system under construction can be analyzed as if it were a single user system, there can be major differences in culture and perception of the product itself, for example between the reporting systems developers and the ultimate end users who actually scrutinize the reports. The persona method [8] was intended to be of help in such situations.

Even though, in a typical study several personas will be identified, there is an implicit assumption in the PSG method that the actions of one persona when using the system have no impact on the actions of other personas. In other words, each persona makes an isolated use of the system. This assumption does not hold in the context of reporting platforms. For example, the actions of a persona responsible for creating a model will most certainly affect the way that a user of the intermediate interface will work subsequently. Thus, for our purposes, i.e. requirements elicitation for reporting platforms in which usage is highly interconnected, there is a special challenge.
4 Applying Persona to the development of reporting using the GREHU system

Our initial assumption was just that the reporting organizations’ division of labor needs to be studied in more detail. Related to this, the organizational and technical solutions should be reviewed based on the findings. Following this, we had an initial plan to use a reporting-specific questionnaire, which includes the following perspectives.

- The amount of reporting activity by the user
- The reporting process
- User’s roles in the reporting process
- Issues about report re-use
- What is an acceptable delay in ad-hoc reporting
- Suggestions on reporting process improvement

Applying the PSG method [2,3], a small set of users was identified to represent the user groups. The selection was fairly random in the sense that when GREHU user organizations were visited for other purposes, the users of the reporting facilities of GREHU in those organizations were given the questionnaire to fill in. The users were interviewed. The interview followed the data that was to be used to form the personas.

In the initial round, 11 people from 3 companies filled in the form. They included people who use operational systems including users responsible for the maintenance of the data, subject-matter experts, and people in managerial positions. From our analysis of the sample data, the people who participated in the survey had a good level of self-awareness concerning their work tasks and their own skill sets.

We classified our sample of users into three groups.

- Group 1. The first group consists of people who just want to improve their knowledge of the GREHU system. These are, typically, not very experienced GREHU users.
- Group 2. The second group is formed by people who know the application area very well but their experience of computing is limited. They may know the GREHU system well and they are aware that it does not completely meet their needs, but they are not yet sufficiently sophisticated as users to propose changes, even those that would be directly to their convenience.
- Group 3. The third group includes people who are sufficiently confident in their understanding of the system actually to make proposals for its development. (Regrettably, there are very few of these people.)

The distribution due to this classification can be seen in Figure 4.

![Figure 4. Distribution of users who responded (The overall number of interviewed users was 11.)](image)

As for the actual wishes that the users propose, it often happens that the change requests may be conflicting and people may disagree about the changes. That is one of the key difficulties in report development. Generally speaking, the users are able to seek improvements in reporting, when they are assisted by a specialist or some other person who has a vision for improvement, but on their own they are unable to verbalize their aspirations. Also, pressure for platform change does not just arise from users’ wish-lits. For example, new legislation can give rise to different regulatory requirements for human resources reporting, necessitating adaptations to the reporting facilities.

In Figure 5 we give an example persona formed based on the data from the interviews.

In terms of the persona method, the picture that emerges from our study is very different from the industry top-down picture. The striking characteristic is that the groups are aspirational. The aspirations can take two forms, Either the user aspires to improve their level of sophistication (in effect moving to one of the other groups) or the user aspires to improving the facilities of the model. By contrast, the conventional wisdom of the major reporting platform vendors gives the impression of a static picture.

The implication of our findings is that the design of the reporting platform should attempt to facilitate the aspirations of users, rather than assume that members of the user community remain in a small number of fixed silos. Existing reporting platforms are designed around major differences between the facilities available to power users and less sophisticated users. Our findings suggest that a more incremental approach to the user interface design would increase user satisfaction.
However, in a sense the results were disappointing. We were clearly expecting more users in the category where the users suggest changes to the system. A potential reason why this did not happen is that the interview was on a too general level, and in fact we would need to explicitly use the questions that we proposed at the beginning of this section to analyze the reporting cycle etc. In short, we need to stimulate the users more on this part.

There are some apparent reasons why this is so particularly in the area of reporting:

- The reporting systems are complex and the users have a hard time understanding the possibilities.
- To fully understand the possibilities in reporting, one also needs to know the underlying data, which is an additional complication.

A further conclusion was that it seemed pointless to complete the persona formation process with the data that we obtained. The existing data illustrates the users' aspirational levels, i.e. high level goals in using the reporting system. It is, however, lack of information on users' usage patterns and behavior patterns, which makes it hard to identify the primary persona(s). The next step will be a re-design of the interview, and then to conduct the interviews anew to obtain better data with an improved data collection stage.

Meanwhile, the persona method includes an important phase to evaluate the proposed features, which is to identify the preference of each user group for features in terms of their usage of those features [2]. Representatives from each user group will be asked about their preference by questionnaire and interviews. Based on the collected data, the user group with the highest coverage of the proposed features can be identified as a primary persona. Due to the limited schedule, our case study has not proceeded so far. However, we found it feasible in the analysis of reporting platforms, as there are already many reporting systems available. It should not be difficult to propose a list of initial features for the platform based on the experience of reporting, and study user's favorable response to the list of features.

5 Conclusions

Our main conclusion is that the classic model of division of labour in business intelligence reporting is very static where in fact the users have aspirations which should be taken into account in the design of information systems. Our other conclusion is that reporting can be a very challenging work area and the user analysis needs highly stimulating questions to get the interesting ideas and views from the users.

Our initial results suggest a model of usage of reporting platforms that differs radically from the model that has suffused the thinking of the major vendors of reporting platforms. In particular, our results draw

<table>
<thead>
<tr>
<th>Name: Mabel Mesa Silva</th>
<th>Sex: Female</th>
<th>Age: 28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Department: Human Resources</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Role: Official in Human Resource Management</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Goals:

- Keep the information updated using the GREHU system.
- Generate reports according to bosses and routine demands.

Skills:

- Knows about the interface of the payment sub-systems.
- Knows about the Human Resources Management and other related issues.
- Has a basic knowledge of computers and other related areas.

Tasks:

- To print out monthly and other reports.
- To input or updated the diary data of the employees.

Wishes:

- To get a better knowledge of GREHU system to make the work easier.

Other Details:

- Keep the information updated using the GREHU system.
- Generate reports according to bosses and routine demands.
- To print out monthly and other reports.
- To input or updated the diary data of the employees.
- To get a better knowledge of GREHU system to make the work easier.
attention to the potentially dynamic aspects of usage over time, given the users’ aspirations. Recent comments by

In the literature on personas, there are frequent references to situations where technology, market and
competition issues are perceived to be more important than the issues that emerge from the use of personas [17].
Although we have no direct evidence of this occurring in the context of reporting platforms, the circumstantial
evidence seems to suggest that this is what has happened in the industry.

It is, of course, clear that the experiment described here was carried out on just one information system
with reporting capabilities with user representatives from just a few user organisations. The reader may object
that the results are just a characteristic of the computing environments in which we conducted our survey.
However, we are confident that a similar picture would emerge of our study were to be replicated in computing
environments using the better-known generic reporting platforms and other types of organisations. This is
backed up by the characteristics of work in the reporting, which logically support our findings. We hope that
further studies on other environments will be obtained to compare them with the results of our study.

References
Products, Proceedings of the 2005 13th IEEE International Conference on Requirements Engineering (RE’05)
282.
December, 2008.
[7] Carroll, J. M. Scenario-based design: envisioning work and technology in system development, John Wiley & Sons,
Inc., New York, NY, 1995
[8] Cooper, A. The Inmates are Running the Asylum. Sams, 1999
July/August, 2001, 58-66
Information and Software Technology 48 (4), 2006
[17] Miller, G., Williams, L. Personas: Moving Beyond Role-Based Requirements Engineering,
http://agile.cse.ncsu.edu/SEMaterials/Personas.pdf
(Business Informatics Research), Kaunas, Lithuania.
Inteligencia de Negocios, Proceedings of XIV Convención Científica de Ingeniería y Arquitectura (CCIA 14),
Havana, Cuba, 2008.

- 159 -