A Comparison of Two Business Rules Engineering Approaches

Lex Wedemeijer
School of Computer Science, Open University, Valkenburgerweg 177, Heerlen, Netherlands
lex.wedemeijer@ou.nl

Keywords: Business Rules, Requirements Engineering, Aspect Oriented Modelling, Modelscope, Relation Algebra, Ampersand.

Abstract: We compare two contemporary approaches under development for business rules engineering with the aim to understand their coverage of business rules and their potentials for requirements engineering. One approach, Aspect Oriented Modeling, focuses on events, state transitions and the synchronization of transitions between objects. The other approach, Ampersand, focuses on invariant rules that should be complied with regardless of the business events taking place. Our comparison brings out that either method can adequately capture some types of business rule, but not others. We conclude that a combination of the two approaches may be a significant contribution to the methods and tools for business rules engineering currently available.

1 INTRODUCTION

All businesses operate according to rules. The rules, whether formally acknowledged or tacitly assumed, influence and control the behaviour of the business. Various approaches to capture, model, implement and enforce business rules exist (zur Mühlen and Indulska, 2010). Still, which approach toward requirements engineering for business rules is best suited to what business environment is open for debate. This paper looks at two contemporary methods and tools under development. The first method is called Aspect Oriented Modelling, or AOM, and it comes with a tool ModelScope (McNeile and Roubtsova, 2010). The other method and tool is called Ampersand (Joosten, 2007). We compare the two by taking the leading example put forward by proponents of one method, and redevelop that example using the opposite method and toolset.

Our aim is to learn and understand about the coverage of business rules and potential for requirements engineering, outlining major differences and semantic issues that we encountered. Doing so may provide useful insights to method engineers engaged in the creation of new and improved approaches for business rules engineering. However, our intent is not to present a thorough evaluation of the two approaches, that are as yet immature and lack a track record of business engineering projects.

The outline of the paper is as follows.

Section 2 sets the stage. We introduce the notion of Business Rule as the common ground, and we discuss the selection of our two approaches.

Section 3 introduces the Ampersand leading example, which is about an IT Service Desk, and we discuss its redeveloped version using the AOM approach and ModelScope tool. Section 4 describes the fictitious Banking example of the AOM approach and outline its redeveloped version using the Ampersand approach and tool.

Lessons learned from the two translate-and-redevelop exercises are presented in section 5.

Section 6 presents our conclusions. We advocate as a future direction the integration of the two approaches to combine their powers to capture, model, implement and enforce business rules.

2 BUSINESS RULES IN INFORMATION SYSTEMS

A business rule, as defined by the Business Rules Group is "a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business" (Hay, Kolber et al., 2003). Rules can be classified into five broad categories: transformation rules, integrity rules, derivation rules, reaction rules, and production rules (Wagner, 2005). Business rules can and will apply to people, processes, and/or overall corporate behaviour, but in

Wedemeijer L.
A Comparison of Two Business Rules Engineering Approaches.
DOI: 10.5220/0004461701130121
In *Proceedings of the Second International Symposium on Business Modeling and Software Design (BMSD 2012)*, pages 113-121
ISBN: 978-989-8565-26-6

this paper we will only consider rules explicitly captured in some information system of the business.

Business rules in an information system on the one hand, describe and constrain the business data (definitions and recorded data), i.e. the 'terms' and 'facts' referred to in the Business Rules Manifesto (2003). On the other hand, the rules describe, guide and constrain the business behaviour, i.e. the events affecting the information in the system. It requires that all types of event must be described, their operations on data, and it should be clarified whether a particular event should be prevented, discouraged or encouraged according to the rules.

2.1 Selected Approaches

This paper is restricted to two approaches that we selected for three reasons.

First and foremost, each approach is selected for being based on just a single modelling paradigm, and consequently, each is strong in capturing one particular category of business rule as named above.

Second, the approaches are just emerging from the laboratory phase and the comparison may help to improve their fitness for use.

Third, their tools are open source and can be studied in isolation, not being part of some dedicated ERP system, database, or service bus infrastructure.

2.2 AOM and Modelscope Approach

AOM, Aspect Oriented Modelling in full, focuses on transformation rules and the synchronisation of state transitions of business objects. The AOM approach aims to capture object behaviour and synchronize (the composition of) multiple behaviours in the early modelling stages of engineering projects.

AOM relies on Protocol Modelling (McNeile and Simons 2006), an event-based paradigm in which models are composed from behavioural components called Protocol Machines (McNeile and Roubtsova, 2008). By composing and synchronizing behaviour of distinct Protocol Machines, the approach provides "a basis for defining reusable fine-grained behavioural abstractions" (McNeile and Simons 2006).

The tool to support this approach is called ModelScope.

The Modellers' Guide (ModelScope version 2.0, 2004) describes it as "a state transition diagram interpreter which understands how events change the state of objects, what events an object can accept based on its state, and how events create and destroy relationships between objects" (p.6). The idea is that a transition of a business object is accepted only if both its pre- and post-state comply with the rules expressed in ModelScope. Any violation of a rule is detected by the tool. It then either rejects the transition immediately (behaviourtype 'essential') or prompts the user to explicitly allow it (behaviourtype 'allowed'). Either way, the business behaviour is guided towards rule compliance.

ModelScope offers a default user interface to enter data about events, and a facility to provide persistent data storage. The tool also provides callback features for specifying inferences and calculations in java code. These callbacks are invoked whenever an event is presented to the model.

The ModelScope tool is available for free download at: `www.metamaxim.com`

2.3 The Ampersand Approach

Ampersand focuses on integrity rules, or more exactly on what is called invariant rules. These rules are characterized as "agreements on business conduct" that users in the business should comply with at all times. Basically, Ampersand constitutes "a syntactically sugared version of Relation Algebra" (Michels, Joosten et al., 2011). The idea is that the state of the business should at all times comply with the agreements that are expressed as invariant rules in Relation Algebra. Ampersand will determine all the violations of all the rules and report them to the user(s). Obviously, such listings contain derived data only, but instead of being regarded as superfluous data, the Ampersand approach views these violations as triggers. Each rule violation constitutes a signal to the user that work must be done to remedy the problem, thereby guiding the business behaviour towards rule compliance.

Ampersand is also the name of a software tool to compile scripts written in the Ampersand vernacular. The tool comes with a facility to provide persistent data storage, and a user interface can also be specified for data entry. The tool can present a diagram of the Conceptual Model complete with its populations and rule obeisances. Moreover, it can also produce an extensive functional-specifications document.

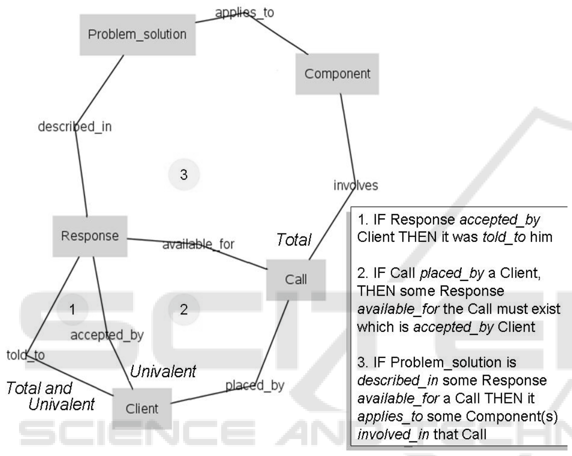
The Ampersand tool is available for free download via the SourceForge community at:
`www.sourceforge.com/ampersand`

3 AMPERSAND TO AOM

This section describes the characteristics of the Ampersand leading example, and outlines how this example is redeveloped according to the AOM approach and using the ModelScope tool. The challenge is to translate the invariant rules from Ampersand into corresponding events with synchronized state transitions in ModelScope.

3.1 IT Service Desk in Ampersand

The leading example for Ampersand is the IT Service Desk. The Conceptual diagram, with 5 concepts and 7 relations, is depicted in figure 1. The example lists several business rules. There are several multiplicity constraints that concern one relation only. The rules marked 1, 2 and 3, involve more than just one relation. In Ampersand, these are known as cyclic rules because the relations involved in them can be seen to form a cycle in the diagram.



1. IF Response *accepted_by* Client THEN it was *told_to* him

2. IF Call *placed_by* a Client, THEN some Response *available_for* the Call must exist which is *accepted_by* Client

3. IF Problem_solution is *described_in* some Response *available_for* a Call THEN it *applies_to* some Component(s) *involved_in* that Call

Figure 1: Conceptual diagram for Service Desk example.

The basic Ampersand script for this example contains some 60 lines of code. This example was translated into a ModelScope script of 500 lines of code, plus 20 components with callbacks of some 25 lines of java code each. These counts are slightly unfair though, because the Ampersand script lacks the user interface specifications that had to be added to the ModelScope script.

3.2 Translating to the AOM Approach

Findings from our translation effort are explained in this section. The main point is that we find AOM to be not very well suited to capture the invariant rules which are of prime importance in Ampersand.

In redeveloping the above example as a ModelScope model, we decided to translate each concept, and also each relation as a distinct object. The argument for not capturing the Ampersand relations as ModelScope relations will be explained below.

3.2.1 Integrity Rules

A major difference between Ampersand and AOM was immediately encountered. The Relational paradigm presumes two basic rules: Entity integrity and Referential integrity. Entity integrity dictates that every customer has a name that is unique among all customers. To contrast, the Object-Oriented paradigm does not assume such a rule, and ModelScope does permit different customers to have identical names. There is a demand that "the combination of the OID and machine-type properties of a machine must be unique in a system" (McNeile and Simons 2006) but as the system itself (i.e. ModelScope) creates and controls the OID's, the demand is trivial for the system, but not for the users. We had to make explicit provisions in the ModelScope model to capture the integrity rules:

– Entity Integrity is captured by rules that prohibit the insertion of an object instance whenever its name is either blank, or already present in that concept. Moreover, we also made sure to prevent duplicate tuples, i.e. we prohibit insertion of any tuple already present in the relation.
– Referential Integrity is safeguarded by enforcing certain events, not rules. In the user interface, we ensure that the user can insert a tuple only by selecting one instance of the source object and one instance of the target object. In addition, we provided Cascading Delete events to supersede the default Delete events that the user interface provides.

3.2.2 Lifecycle Support

Another difference between AOM and Ampersand is in their dealing with object life cycles, specifically in dealing with the end of the object life cycle. In Ampersand, a tuple or atom that no longer records relevant data about the real world is deleted from the data store. But in ModelScope, object data is never removed from storage; a tool feature that is deliberately incorporated to keep track of the state and the dynamics of any object instance at all times.

The designer however may introduce an object state 'deleted' and ensure that instances in this state will accept no events. Like integrity, this subtle difference also requires considerable attention when translating from Ampersand to AOM.

3.2.3 Multiplicity Rules

Multiplicity rules are not hard to implement in ModelScope. Two distinct behaviours, named "Univalent" and "Total", are included in the specification of the designated relations. For ease of use, and for consistency with Ampersand, we assigned both behaviours the 'allowed' behaviourtype, i.e. ModelScope will signal any transition attempting to violate univalence or totality, but the user can override the signal and allow the transition.

Univalence can be handled per tuple of the relation. This behavioural component produces a signal whenever a tuple is inserted but the tuple's source object instance happens to be already represented in another tuple of the same relation.

Total is slightly harder to deal with. A violation of the Total multiplicity cannot be handled per tuple of the relation, but the entire extension of the source object has to be inspected. Even more: the insertion of a new instance of the source object must be allowed even though it means violating the cardinality rule on all relations specified as Total. This is because prohibiting the insertion of source data would effectively halt all data processing.

The Ampersand example happens to have no relations with injective or surjective cardinalities, but these might have been provided for in much the same way. Homogeneous-relation properties such as reflexive, symmetric or transitive, may also be provided for without much ado.

3.2.4 Why not Capture Relations as Relations

We indicated how we translated Ampersand relations to ModelScope objects, not relations. Our reason for doing so is that in ModelScope, the notion of 'relation' is restricted to functional relations only, in keeping with Object-Oriented approaches (Booch, Jacobson et al., 1997). That is, the tool enforces univalence but not referential integrity for its relations. In Ampersand however referential integrity holds rigidly for all relations, but univalence is an option. Turning the Ampersand relations into ModelScope objects allowed us to safeguard the rules that Ampersand had imposed on the various relations.

3.2.5 Rules Involving more than One Relation

Three rules in the Ampersand leading example involve more than one relation. Basically, all three rules are inclusion assertions. The first one is just a basic set inclusion (remember that by definition, a relation is a special kind of set). The two other rules use the relational composition operator, perhaps better known as natural join (Codd 1970).

To translate these rules from Ampersand to ModelScope, they must be written as rules for state transition of business objects, and we must ensure their proper synchronisation. We therefore need to consider all state transitions that might violate the rule; and also those that undo an existing violation.

We decided to create in ModelScope one behavioural component (object type) for each rule. In order to understand the impact of various behaviourtypes for this object, we went so far as to implement separate versions with distinct behaviourtypes for each rule. In doing this, we in fact merged a Business Data Model and an uncoupled Business Rules Model into a single ModelScope script.

For rule number 1, two transitions may produce a violation: the addition of an *accepted_by* tuple, and deletion of a *told_to* tuple. Likewise, a violation can be remedied in two ways: by adding a tuple in the former relation, or deleting one from the latter.

Whenever a composition of relations is involved, the number of state transitions that potentially cause or remedy a violation is multiplied. More complicated rules for relations will require ever more complex callback code to assess the effects of a single event or state transition being presented to the model. Indeed, the java callback code becomes prohibitively complex even for invariant rules of moderate size.

3.2.6 Once a Violation has been Allowed

The focus of Ampersand on invariant rules means that it reports on the rule violations that it detects in the persistent data. However, there is no notion of persistent violations in AOM, and the ModelScope tool does not provide any options to assess the stored data for static violations. Still, a report of rule violations in ModelScope similar to Ampersand can be produced, but it takes some tinkering. This is because rule violations are in fact derived data instances, and the ModelScope tool is not well suited to deal with this kind of derived data.

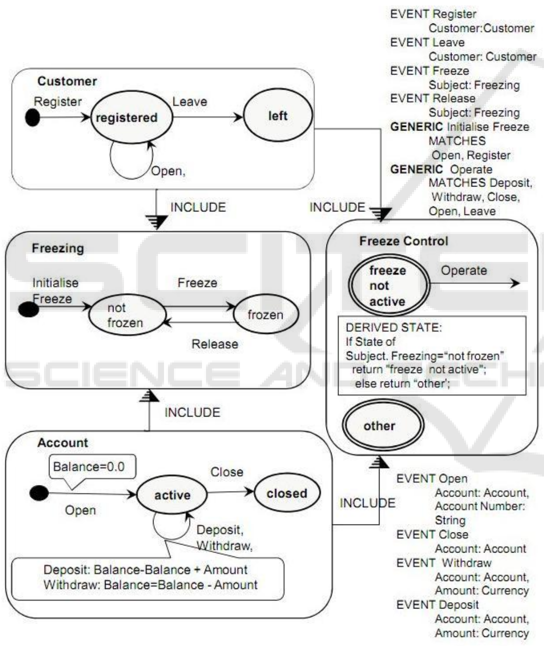
4 AOM TO AMPERSAND

The previous section described the translation of a leading Ampersand example into a working example for AOM. In this section we attempt the opposite direction: we take a fictitious Banking example that is leading for the AOM approach, and we outline how it may be redeveloped using the Ampersand approach and tool. The challenge is to translate the events and synchronized state transitions from ModelScope into a corresponding set of rules in Ampersand.

4.1 Banking Example

A leading example for AOM is called Banking. A diagram of behavioural components is depicted in figure 2, leaving out some components such as Address, Savings-Account and Transfer. The example covers various events to be synchronized, such as opening or closing a bank account, making deposits or withdrawals, guarding against overdrawing an account, and temporarily freezing an account in order to prevent withdrawals.

The example consists of a ModelScope script of about 100 lines of code, plus a dozen callback components with between 5 and 25 lines of java code. The Ampersand script after translation has 50 lines of code. Again, the count is unfair because Ampersand proved inadequate to capture all the rules, and we omitted details of the user interface specification.



```
EVENT Register
        Customer:Customer
EVENT Leave
        Customer: Customer
EVENT Freeze
        Subject: Freezing
EVENT Release
        Subject: Freezing
GENERIC Initialise Freeze
        MATCHES
         Open, Register
GENERIC  Operate
        MATCHES Deposit,
         Withdraw, Close,
        Open, Leave
```

```
DERIVED STATE:
If State of
Subject. Freezing="not frozen"
  return "freeze  not active";
  else return "other";
```

```
EVENT Open
        Account: Account,
        Account Number:
        String
EVENT Close
        Account: Account
EVENT  Withdraw
        Account: Account,
        Amount: Currency
EVENT Deposit
        Account: Account,
        Amount: Currency
```

```
Deposit: Balance-Balance + Amount
Withdraw: Balance=Balance - Amount
```

Figure 2: Diagram for Banking example.

4.2 Translating to Ampersand

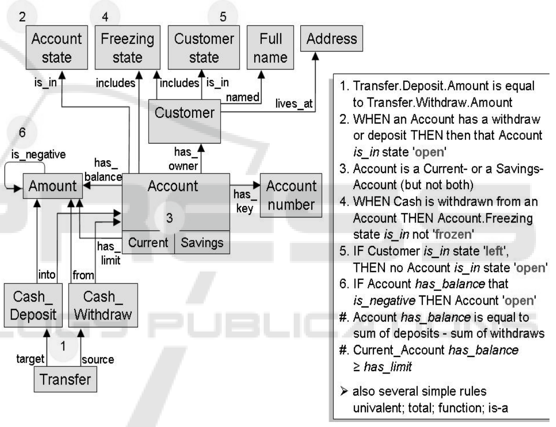
The main issues encountered in the translation are explained in this section. We find the invariant rules of Ampersand to be excellently suited to capture several of the state transition rules in the example. However, various other rules embedded in the state transitions failed to be captured by Ampersand.

4.2.1 Determining Concepts and Relations

The diagram above is compliant with the AOM approach, but not with the Relational paradigm that Ampersand is based on. To create a working Ampersand script, several adjustments had to be made. For one, we omitted various derived attributes, states and behaviours from the AOM model. Furthermore:

– regular attributes such as the customers' address, and even the attributes that act as user-supplied identifying names (Accountnumber, Full-name), became concepts in their own right,
– as each behaviour-state is recorded by way of an attribute with a pre-assigned (hard-coded) range of possible values, we translated all STATES attributes into distinct concepts, except for those behaviours that have only a single state,
– three events in the AOM model carry an important data item: the amount of the cash deposits and withdrawals. This data must be recorded in the Ampersand model, and we added the appropriate concepts and relations.

Figure 3 is the translated Conceptual diagram.



1. Transfer.Deposit.Amount is equal to Transfer.Withdraw.Amount
2. WHEN an Account has a withdraw or deposit THEN then that Account *is_in* state 'open'
3. Account is a Current- or a Savings-Account (but not both)
4. WHEN Cash is withdrawn from an Account THEN Account.Freezing state *is_in* not 'frozen'
5. IF Customer *is_in* state 'left', THEN no Account *is_in* state 'open'
6. IF Account *has_balance* that *is_negative* THEN Account 'open'

#. Account *has_balance* is equal to sum of deposits - sum of withdraws

#. Current_Account *has_balance* ≥ *has_limit*

➢ also several simple rules univalent; total; function; is-a

Figure 3: Conceptual diagram for Banking, translated.

4.2.2 Determining the Rules

Rules in the AOM example are expressed as constraints on state transitions, with the aim to guarantee that the data captured in the model cannot enter states that have no counterpart in reality.

We expected that all of the state transition rules in the AOM example would translate into non-cyclic rules on relations in Ampersand. In general, non-cyclic rules are expressed by way of one composite relation and specific (named) states of concepts, such as 'opened' or 'active', their most common type being multiplicity constraints. Our expectation proved to be correct, with just a few exceptions.

4.2.3 Implicit Capture of Cyclic Rule

When transferring money between accounts, the withdrawn amount is assumed to be exactly equal to the deposited amount. This requirement is implemented implicitly, by having only one "amount" attribute in the "Transfer" event, to be used in both the Cash withdrawal and Cash deposit. We think that equality ought to be made explicit as a business rule.

Likewise, when money is transferred, the two accounts are assumed to be different. However, we found that the ModelScope script did not intercept transfers into the same account. The event was accepted, but resulted in an unexplained error.

4.2.4 State Transitions Capture Invariant Rule

The "Customer" object comes with three possible states: 'registered', 'pending leave', and 'left'. The Modellers Guide explains that before a customer leaves the Bank, all of the accounts must be closed. While the procedure to close the accounts is running, the customer's state is labelled as 'pending leave'.

So the general idea is that a left Customer has no open accounts. This is a splendid example of an invariant rule. However, ModelScope does not capture this rule. Instead, it implements constraints on state transitions simulating the process of a customer leaving. But: once a customer has left, ModelScope still accepts certain events for that customer, such as (un)freezing the status, or even depositing cash into his accounts that suddenly re-open. We attribute this imperfection to complexity: the number of state transitions to be accounted for rapidly rises as the number of objects, behaviours and events increases.

4.2.5 Immutable Property

For some attributes or object states, no transition is specified, e.g. the customer's Full Name, or a customer who has status 'left'. By implication, the attribute value or state will be for ever immutable. A corresponding immutability-rule might be specified, but we did not do so because Ampersand does not support temporal rules nor pre- and post-states.

5 LESSONS TO BE LEARNED

Lessons to be learned from these two translate-and-redevelop exercises are presented in this section. We point out basic differences between the approaches, and some weaknesses in the current tool support for each approach are pointed out.

5.1 Paradigm Mismatches

Both approaches support business rule engineering, but using fundamentally different paradigms.

5.1.1 Object-orientation vs Relation Algebra

A prime difference is the modelling paradigm underlying each approach: OO versus Relation Algebra. As a result, basic issues such as identity, entity and referential integrity are treated widely different as was discussed above.

5.1.2 Instance- vs Set-oriented Rules

A fundamental difference between AOM and the Ampersand approach is that AOM is designed for synchronization of events and state transitions on individual object instances, i.e. it implements transition rules. Such rules are expressed and evaluated on the basis of single instances following a fine-grained behavioural analysis.

Ampersand however is geared towards invariant rules expressed by assertions on entire relations. This is a very compact and powerful way to specify requirements on a large-scale, structural level. The downside is that, if a rule involves many relations, the details are lost of how a single state transition may incur one or multiple rule violations.

5.1.3 Dealing with a Rule Violation

Both AOM and Ampersand acknowledge that in a working business environment, a rule sometimes needs to be violated. But here again, the approaches are fundamental different.

The AOM approach focuses on state transitions, and rules are maintained by permitting or forbidding object transitions depending on pre- and post-states of the objects involved in the event. If ModelScope determines that some state transition rule is violated, the event is either rejected offhand, or it is presented to the user for acceptance (behaviourtype 'allowed'). But once accepted, the data is stored. Thus, violation is regarded a dynamic, temporary phenomenon.

The Ampersand tool does not deal with transitions or pre- and post-states. It inspects the stored data and calculates the (static) violations of the rules. These are reported to the user, leaving it to the user to assess the errors in the data and make amends. The tool is not concerned with the particular events, or chain of events, that may have caused the violations.

5.2 Unresolved Issues

Apart from the fundamental mismatches mentioned above, we also noticed several issues that need attention in both approaches.

5.2.1 Workflow Support

It is a matter of opinion (Hofstede, van der Aalst, et al., 2003) whether workflow specifications constitute business rules, or whether they are just a way to implement the underlying, more fundamental business rules.

ModelScope takes a somewhat ambivalent position by providing a behaviourtype 'desired'. It is expressed only in the user interface, where it indicates to the user which next step (event or transition) is desired for an object instance, when that instance happens to be on display. The engineer may apply this behaviourtype, but the tool provides no proper guidance or control. There is no link to some encompassing workflow; nor is it clear how to go about if different workflows desire conflicting state transitions for a particular behaviour.

Ampersand takes the position that a workflow is merely an implementation, one possible chain of actions that a user may undertake to remedy violations and comply with all the invariant rules. Therefore, Ampersand supports the user in remedying violations, but it offers no features to support detailed workflow design.

5.2.2 Derived Data

In practice, there is lots of derived data about, either stored as persistent data or used on the fly in the software code. Rules to control such data are sometimes called 'projector' rules (Dietz, 2008). It is a rule of thumb that derived data ought to be eliminated from data models, but we encountered several types of derived data in the examples.

One subtype of derived data is derived attribute value, such as the balance of a bank account. ModelScope can handle this kind of data very well by way of java callback routines. Ampersand however, relying on Relation Algebra only, does not deal very well with derived values.

However, another subtype is derived existence (or demise) of some instance of a concept or relation. In particular, we may consider every rule violation to be just such a derived instance. Here, the situation is reversed: Ampersand is well suited to handle such instances, but ModelScope can hardly deal with derived object instances or life cycles.

5.3 Shortcomings of the Tools

Both the Ampersand and ModelScope approaches, and their tools, are under development. Our translation efforts brought out various points of lacking user functionality, frustrating our goal of comparising the two approaches. Such shortcomings can well be mended in upcoming tool releases.

5.3.1 ModelScope Tool

The ModelScope interface offers a NAME attribute for the user to identify objects. However, the tool ignores these identifiers and uses internal object-identifiers instead. This is a cause of confusion whenever the user makes a mistake with identifiers, such as accidentally entering the same identifying name twice, which is accepted by the tool without signalling the duplicate.

Another issue that needs attention is derived data and code redundancy. It was mentioned how a compound business rule will involve more than one relation. Hence, every transition on any of those relations has to assess the same rule. The implication is that either the program code for that rule has to be duplicated for each transition (with a certain risk of becoming inconsistent if the code is edited), or some derived data object must be designed in order to encapsulate the code. ModelScope provides little support for either solution.

Regrettably, the current tool version does not generate diagrams or specifications for end users or engineers to ease their understanding, or to help in reviewing of the set of objects and events after completion. Such documentation would be helpful particularly if models grow in size and the numbers of events that must be synchronized increase.

5.3.2 Ampersand Tool

The current implementation of Ampersand cannot deal with rules that involve numeric calculations or comparisons, nor timestamps and date intervals. As a result, important business rules defy to be implemented, such as the basic rule that the balance of a bank account equals the sum of deposits minus the sum of withdrawals, or the rule that the negative balance of a Current account shall never be lower than the limit set for it.

A further drawback of the current Ampersand version is that it currently lacks a proper user interface for entering and editing data.

6 CONCLUSIONS

We conducted a detailed comparison of two emerging approaches and tools for modelling business rules. The first one is called Aspect Oriented Modelling (AOM) and comes with a tool named ModelScope. The other approach and tool is called Ampersand. By comparing these two, we aimed to learn and understand about the impact and consequences brought about by the choice of modelling approach underlying each method.

6.1 Conflicting Approaches

The two approaches were selected for focusing on one particular category of rules. The AOM approach is strong in transformation rules, and Ampersand is strong in integrity (invariant) rules.

Our comparison efforts show the negative consequences of this: AOM is weak in dealing with invariant rules, especially compound ones. And Ampersand provides poor support for state transition rules. Moreover, our translate-and-redevelop exercises disclose semantic issues about each method that were not previously noticed. The two approaches are conflicting in important aspects:

– in the modelling paradigm, relational or object-oriented, causing engineers to produce fundamentally different models,
– in the instance- or set-oriented perspective to be taken for rules, and
– in how a violation is dealt with.

The implication is that prior to conducting an actual analysis of business rules, a business engineer must decide which approach is most suited to the case at hand. If the case at hand is primarily concerned with the proper processing of events, and synchronization of dynamic state transitions for multiple objects, then the AOM-approach is at an advantage. A point of concern then is the size of the model, and the number of state transitions to be accounted for. If the focus is more on the static states of large data collections, and compliance to invariant rules, then the Ampersand approach seems to be more appropriate.

In our view, the two approaches provide as yet inadequate support for requirements engineering. This may be no surprise as both approaches, and their support tools, are still under development. The current state of affairs is that both approaches have serious drawbacks, and one approach is not superior to the other. At present, neither supports all needs of developers engaged in business rules engineering.

6.2 Limitations of the Comparison

In our comparison, we looked at only two small examples of business context. A thorough evaluation of the approaches would require a comparison of quality, flexibility, maintainability and other features of the delivered systems designs. But as the approaches and their ways of working are just emerging from the laboratory phase, no realistic operational models and implementations were available to be scrutinized and compared in our comparison.

Moreover, we selected the approaches for being based on just a single modelling paradigm. We found little support in either approach for the other categories of business rules: workflow (reaction and production rules), and derivation rules. From this, it may be speculated that other approaches that target a single modelling paradigm and a single category of business rules, will also fall short in providing adequate support for rule engineers in operational business environments.

6.3 Direction for Development

We stipulate that approaches may well be combined to augment one another. This is because system engineering efforts generally involve aspects of large-scale structural requirement analysis as well as fine-grained behavioural analysis.

Ampersand enables to express invariant rules that are attractively simple, yet have a wide impact, as a single rule can encompass multiple relations each with extensive populations. These features are needed in early stages of requirements engineering, when overall structure and consistency is the issue, not detail.

The AOM approach is strong in the modelling of events and synchronization of multiple behaviours, important features in later stages of engineering, when precise details are at stake.

An engineering approach and companion tool combining the expressive power of invariant rules of Ampersand with the detailed capabilities of controlling state transitions of AOM, would significantly contribute to the field of business rules engineering. It would permit to capture the invariant rules having a large scope in the early stages of engineering, whereas fine-grained rules for state transitions and workflow rules could be added later on. We expect that it is possible to reduce the size of the overall models and to keep the number of state transitions moderate. Combining the capabilities would reduce the drawbacks of either approach, and provide a more complete coverage of the engineering needs, which in turn would result in a higher quality of deliverables. Work to bring event synchronization capabilities to the Ampersand metamodel has begun, but it has not resolved all the fundamental differences that must be overcome in the conflicting approaches (Roubtsova, Joosten et al, 2010).

The final word has yet to be said for the best approach and tool, depending on the kind of business environment, to supporting business rules and requirements engineering.

ACKNOWLEDGEMENTS

We wish to thank the organizers and reviewers of the symposium on Business Modeling and Software Design for their helpful comments on an earlier version of this work. We also thank dr. Roubtsova and prof. Joosten for their assistance with the Modelscope and Ampersand toolsets.

REFERENCES

Booch G, Jacobson I, Rumbaugh J. Unified Modeling Language, Rational Software Corporation, version 1.0, (1997)

Business Rules Manifesto (2003). At: www.business rulesgroup.org/brmanifesto.htm. Version 2.0. Edited R.G. Ross. Last accessed 24 march 2012

Codd EF. (1970). A relational model of data for large shared data banks. In: *Communications of the ACM* 13(6): 377-387.

Dietz JLG. (2008). On the Nature of Business Rules. In: *Advances in Enterprise Engineering* eds. Dietz, Albani and Barjis, Springer Berlin Heidelberg. 10: 1-15.

Hay JD, Kolber A, Anderson Healy K. (2003) *Defining Business Rules - what are they really*. Final report. BusinessRulesGroup. At: www.businessrulesgroup .org/first_paper/BRG-whatisBR_3ed.pdf. Last accessed 24 march 2012

Hofstede A ter, Aalst W van der, et al. (2003). Business Process Management: A Survey. In: *Business Process Management*. M. Weske, Springer Berlin / Heidelberg. 2678: 1019-1019.

Joosten S. (2007) Deriving Functional Specification from Business Requirements with Ampersand. At: icommas .ou.nl/wikiowi/images/e/e0/ampersand_ draft_2007nov.pdf. Last accessed 24 march 2012

McNeile A, Simons N. (2006) Protocol modelling: A modelling approach that supports reusable behavioural abstractions. In: *Software and Systems Modeling* 5(1): 91-107.

McNeile A, Roubtsova E. (2008). CSP parallel composition of aspect models. In: *Proceedings of the 2008 AOSD workshop on Aspect-oriented modeling*. Brussels, Belgium, ACM: 13-18.

McNeile A, Roubtsova E. (2010). Aspect-Oriented Development Using Protocol Modeling. In: *Transactions on Aspect-Oriented Software Development* VII. S. Katz, M. Mezini and J. Kienzle, Springer Berlin / Heidelberg. 6210: 115-150.

Michels G, Joosten Se, van der Woude J, Joosten S. (2011). Ampersand, Applying Relation Algebra in Practice. In: *RAMICS 2011, Relational and Algebraic Methods in Computer Science*. Rotterdam, The Netherlands, May 30-June 3. H. de Swart, Springer Berlin / Heidelberg. LNCS 6663: 280-293.

ModelScope Version 2.0 (2004), *Modellers' Guide Version 15*. At: www.metamaxim.com. Last accessed 24 march 2012

Roubtsova E, Joosten S, Wedemeijer L. (2010) Behavioural model for a business rules based approach to model services. June 2010. In: *BM-FA '10: Proceedings of the Second International Workshop on Behaviour Modelling: Foundation and Applications*.

Wagner G. (2005). Rule Modeling and Markup. In: *Reasoning Web*. eds. N. Eisinger and J. Maluszynski, Springer Berlin / Heidelberg 3564: 96-96.

zur Mühlen M, Indulska M. (2010). Modeling languages for business processes and business rules: A representational analysis. In: *Information Systems* 35(4): 379-390.