4.8.4 Zahlungsbedingungen

Die Prüfung, ob Zahlungsbedingungen (Zahlungseingang, Gesamtbetrag erhalten, Skontobedingung) erfüllt sind, wird in der Java-Regel org.nuclet.mt940.rule.CheckBankTransactionRef realisiert. In dieser Regel ist die Priorisierung der zu prüfenden Zahlungsbedingungen festgelegt, und es wird bei Erfüllung der Zahlungsbedingungen ein Statuswechsel auf dem Referenzobjekt (bzw. den Referenzobjekten) initiert.

Mehr zum abgestuften Prüfverfahren, dass standardmäßig implementiert ist, ist im Abschnitt 2 zu finden.

org.nuclet.mt940.rule.CheckBankTransactionRef

```
package org.nuclet.mt940.rule;


import java.util.ArrayList;
import java.math.BigDecimal;
import java.util.Calendar;
import java.util.Collections;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.List;

import org.nuclos.api.annotation.Rule;
import org.nuclos.api.businessobject.BusinessObject;
import org.nuclos.api.context.RuleContext;
import org.nuclos.api.context.UpdateContext;
import org.nuclos.api.exception.BusinessException;
import org.nuclos.api.provider.BusinessObjectProvider;
import org.nuclos.api.provider.QueryProvider;
import org.nuclos.api.provider.StatemodelProvider;
import org.nuclos.api.rule.UpdateFinalRule;
import org.nuclos.api.statemodel.State;

import org.nuclet.common.rule.AbstractRule;

import org.nuclet.mt940.*;
import org.nuclet.mt940.facade.AbstractReferenceFacade;
import org.nuclet.mt940.facade.ReferenceFacadeFactory;
import org.nuclet.mt940.logic.MT940Logic;
import org.nuclet.mt940.statemodel.ReferenceStatemodel;
import org.nuclet.mt940.util.BankTransactionRefComparator;
import org.nuclet.mt940.wrapper.AbstractConditionsOfPaymentWrapper;
import org.nuclet.mt940.wrapper.AbstractReferenceWrapper;

/**
 * @name CheckBankTransactionRef
 * @description Checks if the changes done to the given bank transaction should result in subsequent changes on
its references
 * @usage
 * @change
 *
 *
 * @version 1.4
 * @date 20.02.2014
 * @nuclet org.nuclet.MT940
 * @nucletversion 1.4.0
 * @sincenucletversion 1.0.0
 * @since 11.03.2013
 * @author frank.lehmann@nuclos.de
 *
 */
@Rule(name="CheckBankTransactionRef", description="Checks if the changes done to the given bank transaction
should result in subsequent changes on its references")
public class CheckBankTransactionRef extends AbstractRule implements UpdateFinalRule
{
    private static final BigDecimal BGD_ONEHUNDRED = new BigDecimal("100.0");

    // facades
    protected AbstractReferenceFacade referenceFacade = null;

    // private and protected values
```

```
    private List<State> lstSourceStates = null;
    private State stateDestinationState = null;


    public void initialize(RuleContext context) throws BusinessException
    {
        super.initialize(context);

        this.referenceFacade = ReferenceFacadeFactory.getReferenceFacade();

        if (this.referenceFacade == null) {
            throw new BusinessException("Method getReferenceFacade() in org.nuclet.mt940.facade.
ReferenceFacadeFactory does not return a CurrencyFacade instance (940.472.001)!");
        }

        this.lstSourceStates = referenceFacade.getSourceStates(ReferenceStatemodel.StateChange.PaymentReceived);
        this.stateDestinationState = referenceFacade.getDestinationState(ReferenceStatemodel.StateChange.
PaymentReceived);

        if (this.lstSourceStates == null || lstSourceStates.size() == 0) {
            throw new BusinessException("No source states defined in \"" + referenceFacade.getClass()
                + "\" for ReferenceStatemodel.StateChange.PaymentReceived (940.473.001)!");
        }

        if (this.stateDestinationState == null) {
            throw new BusinessException("No destination state defined in \"" + referenceFacade.getClass()
                + "\" for ReferenceStatemodel.StateChange.PaymentReceived (940.473.002)!");
        }
    }

    public void updateFinal(UpdateContext context) throws BusinessException
    {
        initialize(context);

        checkReferences(context);
    }

    /**
     * Check, if the changes done to the given bank transaction should result in further
     * changes and/or state changes of the references <code>BusinessObject</code>
     *
     * @param context the current context
     *
     * @throws BusinessException, in case an error or exception occurs
     */
    private void checkReferences(UpdateContext context) throws BusinessException
    {
        final BankTransaction boBankTransaction = context.getBusinessObject(BankTransaction.class);
        final List<BankTransactionRef> lstReferences = boBankTransaction.getBankTransactionRef();

        for (final BankTransactionRef ref : lstReferences) {
            final Long lngReferenceId = ref.getReferenceId();
            final AbstractReferenceWrapper reference = referenceFacade.getById(lngReferenceId);
            final List<Integer> lstSourceStateNumbers = new ArrayList<Integer>();

            for (final State state : lstSourceStates) {
                lstSourceStateNumbers.add(Integer.valueOf(state.getNumeral()));
            }

            if (lstSourceStateNumbers.contains(reference.getNuclosStateNumber())) {
                Date datPaymentDate = checkFirstPriorityConditions(boBankTransaction, reference);

                if (datPaymentDate == null) {
                    datPaymentDate = checkSecondAndThirdPriorityConditions(reference);
                }

                if (datPaymentDate != null) {
                    final State stateDestinationState = ReferenceFacadeFactory.getReferenceFacade().
getDestinationState(ReferenceStatemodel.StateChange.PaymentReceived);

                    reference.setPaymentDate(datPaymentDate);
```

```
                    BusinessObjectProvider.update((BusinessObject)reference.getBusinessObject());
                    StatemodelProvider.changeState(reference.getBusinessObject(),
stateDestinationState);
                }
            }
        }
    }

    /**
     * Checks, if the first priority conditions have been fulfilled, i.e.
     *
     * 1a. the reference is marked as "accept first incoming payment"
     *
     * and
     *
     * 1b. an incoming payment has been assigned
     *
     * @param boBankTransaction A <code>BusinessObject</code> of type <code>BankTransaction</code>
     * @param reference An object of type <code>AbstractReferenceWrapper</code>, encapuslating the
     * reference object (client billing, invoice, etc.)
     *
     * @return The payment date, if the reference is marked as "accept first incoming payment" and
     * an incoming payment has been assigned
     *
     */
    private Date checkFirstPriorityConditions(final BankTransaction boBankTransaction, final
AbstractReferenceWrapper reference)
    {
        final Long lngDebitCreditMarkId = boBankTransaction.getDebitCreditMarkId();
        final DebitCreditMark boDebitCreditMark = QueryProvider.getById(DebitCreditMark.class,
lngDebitCreditMarkId);

        if (boDebitCreditMark.getSign().intValue() > 0 && reference.getAcceptFirstIncomingPayment()) {
            return boBankTransaction.getEntryDate();
        } else {
            return null;
        }
    }

    /**
     * Checks, if the second or third priority conditions have been fulfilled, i.e.
     *
     * 2. the reference's total amount has been balanced by incoming payments
     *
     * or
     *
     * 3. the cash discount conditions are met
     *
     * @param reference An object of type <code>AbstractReferenceWrapper</code>, encapuslating the
     * reference object (client billing, invoice, etc.)
     *
     * @return The payment date, if the second or third priority conditions have been fulfilled, i.e.
     *
     */
    private Date checkSecondAndThirdPriorityConditions(final AbstractReferenceWrapper reference)
    {
        final List<BankTransactionRef> lstBankTransactionRef = reference.getBankTransactionRef();
        final AbstractConditionsOfPaymentWrapper conditionsOfPayment = reference.getConditionsOfPayment();
        Calendar calCashDiscountLimit = null;
        Calendar calExtendedCashDiscountLimit = null;
        BigDecimal bgdTotalAmountDiscounted = BigDecimal.ZERO;
        BigDecimal bgdExtendedTotalAmountDiscounted = BigDecimal.ZERO;

        if (conditionsOfPayment != null && conditionsOfPayment.getCashDiscount() != null) {
            // Calculate ...
            // - the discounted amount to be payed and
            // - the time limit for cash discounts,
            //
            // ...in case a cash discount is defined in the conditions of payment...
            //
            bgdTotalAmountDiscounted = reference.getTotalAmountGross().subtract(
```

```
                    reference.getTotalAmountGross().multiply(conditionsOfPayment.getCashDiscount()).divide
(BGD_ONEHUNDRED, 10, BigDecimal.ROUND_HALF_UP)
                );

            if (conditionsOfPayment.getCashDiscountPeriod() != null) {
                calCashDiscountLimit = new GregorianCalendar();
                calCashDiscountLimit.setTime(reference.getDateOfInvoice());
                calCashDiscountLimit.add(Calendar.DAY_OF_YEAR, conditionsOfPayment.getCashDiscountPeriod());

                if (conditionsOfPayment.getCashDiscountExGratiaDays() != null
                    && conditionsOfPayment.getCashDiscountExGratiaDays().intValue() > 0) {
                    calCashDiscountLimit.add(Calendar.DAY_OF_YEAR, conditionsOfPayment.
getCashDiscountExGratiaDays());
                }
            }

            if (conditionsOfPayment.getExtendedCashDiscount() != null) {
                // If an extended cash discount is also given, calculate those values for the extended period
too...

                if (conditionsOfPayment.getExtendedCashDiscountPeriod() != null) {
                    calExtendedCashDiscountLimit = new GregorianCalendar();
                    calExtendedCashDiscountLimit.setTime(reference.getDateOfInvoice());
                    calExtendedCashDiscountLimit.add(Calendar.DAY_OF_YEAR, conditionsOfPayment.
getExtendedCashDiscountPeriod());

                    if (conditionsOfPayment.getCashDiscountExGratiaDays() != null
                        && conditionsOfPayment.getCashDiscountExGratiaDays().intValue() > 0) {
                        calCashDiscountLimit.add(Calendar.DAY_OF_YEAR, conditionsOfPayment.
getCashDiscountExGratiaDays());
                    }
                }

                bgdExtendedTotalAmountDiscounted = reference.getTotalAmountGross().subtract(
                    reference.getTotalAmountGross().multiply(conditionsOfPayment.getExtendedCashDiscount()).
divide(BGD_ONEHUNDRED, 10, BigDecimal.ROUND_HALF_UP)
                );
            }
        }

        if (lstBankTransactionRef != null) {
            Collections.sort(lstBankTransactionRef, new BankTransactionRefComparator());

            Date datPaymentDate = null;
            Date datDiscountedPaymentDate = null;
            Date datExtendedDiscountedPaymentDate = null;
            boolean bHasReferenceAmountBeenBalanced = false;
            boolean bHasDiscountedReferenceAmountBeenBalanced = false;
            boolean bHasExtendedDiscountedReferenceAmountBeenBalanced = false;
            BigDecimal bgdTotalPaymentAmount = BigDecimal.ZERO;

            for (final BankTransactionRef boBankTransactionRef : lstBankTransactionRef) {
                final Long lngOtherBankTransactionId = boBankTransactionRef.getBankTransactionId();
                final BankTransaction boOtherBankTransaction = QueryProvider.getById(BankTransaction.class,
lngOtherBankTransactionId);
                final Long lngOtherDebitCreditMarkId = boOtherBankTransaction.getDebitCreditMarkId();
                final DebitCreditMark boOtherDebitCreditMark = QueryProvider.getById(DebitCreditMark.class,
lngOtherDebitCreditMarkId);

                if (boOtherDebitCreditMark.getSign().intValue() > 0) {
                     bgdTotalPaymentAmount = bgdTotalPaymentAmount.add(boOtherBankTransaction.getAmount());

                     if (!bHasReferenceAmountBeenBalanced &&
                         bgdTotalPaymentAmount.doubleValue() >= reference.getTotalAmountGross().doubleValue()) {
                         bHasReferenceAmountBeenBalanced = true;
                         datPaymentDate = boOtherBankTransaction.getEntryDate();
                     }

                     if (!bHasDiscountedReferenceAmountBeenBalanced &&
                         !BigDecimal.ZERO.equals(bgdTotalPaymentAmount) &&
                         bgdTotalPaymentAmount.doubleValue() >= bgdTotalPaymentAmount.doubleValue()) {
```

```
                        bHasDiscountedReferenceAmountBeenBalanced = true;
                        datDiscountedPaymentDate = boOtherBankTransaction.getEntryDate();
                    }

                    if (!bHasExtendedDiscountedReferenceAmountBeenBalanced &&
                        !BigDecimal.ZERO.equals(bgdTotalPaymentAmount) &&
                        bgdTotalPaymentAmount.doubleValue() >= bgdTotalPaymentAmount.doubleValue()) {
                        bHasExtendedDiscountedReferenceAmountBeenBalanced = true;
                        datExtendedDiscountedPaymentDate = boOtherBankTransaction.getEntryDate();
                    }
                } else {
                    bgdTotalPaymentAmount = bgdTotalPaymentAmount.subtract(boOtherBankTransaction.getAmount());

                    if (bHasReferenceAmountBeenBalanced &&
                        bgdTotalPaymentAmount.doubleValue() < reference.getTotalAmountGross().doubleValue()) {
                        bHasReferenceAmountBeenBalanced = false;
                        datPaymentDate = null;
                     }

                    if (bHasDiscountedReferenceAmountBeenBalanced &&
                        !BigDecimal.ZERO.equals(bgdTotalPaymentAmount) &&
                        bgdTotalPaymentAmount.doubleValue() < bgdTotalPaymentAmount.doubleValue()) {
                        bHasDiscountedReferenceAmountBeenBalanced = false;
                        datDiscountedPaymentDate = null;
                    }

                   if (bHasExtendedDiscountedReferenceAmountBeenBalanced &&
                        !BigDecimal.ZERO.equals(bgdTotalPaymentAmount) &&
                        bgdTotalPaymentAmount.doubleValue() < bgdTotalPaymentAmount.doubleValue()) {
                        bHasExtendedDiscountedReferenceAmountBeenBalanced = false;
                        datExtendedDiscountedPaymentDate = null;
                    }
                }
            }

            if (bgdTotalPaymentAmount.doubleValue() >= reference.getTotalAmountGross().doubleValue()) {
               return datPaymentDate;
            } else if (bgdTotalPaymentAmount.doubleValue() < reference.getTotalAmountGross().doubleValue()
                      && datDiscountedPaymentDate != null && calCashDiscountLimit != null
                      && datDiscountedPaymentDate.before(calCashDiscountLimit.getTime())) {
                return datDiscountedPaymentDate;
            } else if (bgdTotalPaymentAmount.doubleValue() < reference.getTotalAmountGross().doubleValue()
                      && datExtendedDiscountedPaymentDate != null && calExtendedCashDiscountLimit != null
                      && datExtendedDiscountedPaymentDate.before(calExtendedCashDiscountLimit.getTime())) {
                return datExtendedDiscountedPaymentDate;
            } else {
                return null;
            }
        } else {
            return null;
        }
    }
}
```