Customizing und Integration in der Oracle Sales Cloud

Wassili Billon/Yves Chassein
PROMATIS software GmbH
Hamburg/Ettlingen

Schlüsselworte

#OracleSalesCloud
#SAP
#Integration
#ExtensionMitApplicationComposer

Einleitung

Zentrale Problemstellungen bei der Einführung von Standardsoftware in der Cloud, bedingt durch eine Reihe von Limitierungen im Vergleich zu klassischen On-Premises-Implementierungen, sind das Customizing und die Integration mit Fremdsystemen. Dies gilt speziell auch für die Oracle Sales Cloud. Der Vortrag zeigt auf, welche Möglichkeiten und Herausforderungen es diesbezüglich in der Oracle Sales Cloud gibt. Anhand folgender vier Fallbeispiele aus einem realen Kundenprojekt wird der flexible Einsatz des Oracle Sales Cloud Application Composers behandelt:

1. Integration zu SAP.

2. Aufbau eines Genehmigung-Workflows für finanzkritische Attribute am Kundenstamm.

3. Aufbau eines Mandanten-Konzepts, um Stamm- und Bewegungsdaten zwischen verschiedenen Organisationen abzugrenzen.

4. Aufbau eines Audit-Logs, um alle Veränderungen an den CRM Objekten „Account, Contact, Lead und Opportunities“ zu verfolgen.

Die ausgewählten Fallbeispiele stellen Anforderungen dar, die in vielen CRM-Projekten eine hohe Priorität besitzen. Vor diesem Hintergrund wird aufgezeigt, inwiefern die Oracle Sales Cloud in der Lage ist, diese Anforderungen umzusetzen, ohne sich gleichzeitig vom Paradigma der Standardsoftware zu entfernen. Hierbei geht es insbesondere um das Spannungsfeld zwischen dem Wunsch, eine stets releasefähige Standardsoftware einzuführen und der Möglichkeit, mit Hilfe des Oracle Sales Cloud Application Composers die Anforderungen der Anwender umzusetzen.

Ausgangssituation

Wir gehen von folgendem Business Case aus: Die Burger GmbH setzt aktuell SAP als ERP- sowie CRM System On-Premise ein. Der Kunde möchte sich IT-strategisch neu aufstellen und hat sich entschieden seine CRM-Anwendungen und -prozesse in die Cloud zu verlagern. Im Rahmen des Auswahlprozesses ist seine Wahl auf die Oracle Sales Cloud gefallen. Der Rest seiner Business Prozesse soll aber weiterhin in SAP abgewickelt werden. Aus diesem Grund benötigt er seine Kundenstammdaten in beiden Systemen. Es wurde entschieden, dass die Oracle Sales Cloud das führende System für die Kundenstammdaten, Kontakte und Opportunities ist. Somit müssen Änderungen oder Neuanlagen von Datensätzen in die SAP Applikation übertragen werden.

Das Geschäftsfeld der Burger GmbH trennt sich in zwei Töchter auf, die *Fleisch und Wurst GmbH* sowie die *Vegetarisch und Vegan GmbH*. Beide Töchter sollen autark voneinander arbeiten können und dürfen keinen Zugriff auf die CRM Objekte (Kunden, Kontakte, Leads, Opportunities) in der Oracle Sales Cloud des jeweils anderen haben.

Integration zu SAP

Um Aktualisierungen und Neuanlagen in der Oracle Sales Cloud in die SAP Applikation zu synchronisieren, wurde entschieden die Oracle SOA Suite als Middleware einzusetzen. Diese soll als Mittelsmann zwischen der Oracle Sales Cloud und SAP fungieren. Um die Integration umsetzen zu können, müssen zuerst drei wichtige Fragen geklärt werden:

- Soll die Übertragung periodisch oder Realtime durchgeführt werden?
- Gibt es einen Rückgabewert bei der Übertragung?
- Sollen die Objekte komplett übertragen werden, oder nur die geänderten Werte?

Bei der Burger GmbH soll eine direkte Übertragung stattfinden, die aber asynchron zum Speicherprozess laufen darf. Als Rückgabewert der Anlage muss die SAP Kundennummer in die Oracle Sales Cloud zurück geschrieben werden. Es soll immer das gesamte Objekt übertragen werden.

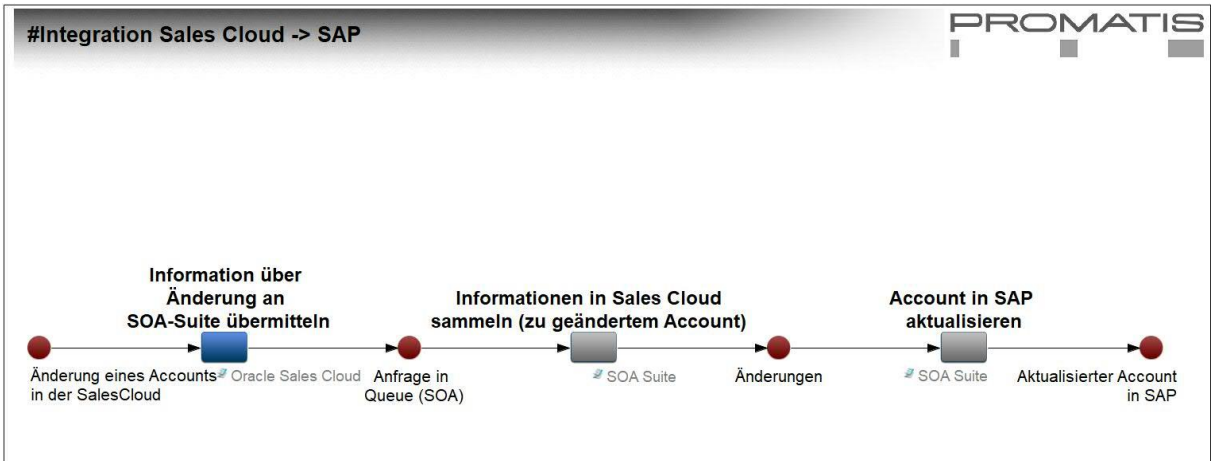

Abbildung 1: Integration Oracle Sales Cloud und SAP

In der Oracle Sales Cloud wurde die Umsetzung auf Grundlage von Triggern, Groovy Funktionen und einem Custom Object durchgeführt. Wenn eine Änderung in der Oracle Sales Cloud gespeichert wird, feuert ein „*Before Update in Database*"-Trigger. Dieser Trigger erstellt über eine Groovy Funktion in ein Custom Object einen neuen Eintrag. Dieser Eintrag repräsentiert den geänderten Kunden. Für jeden Eintrag in dem Custom Object wird über ein „*After Insert*"-Trigger ein Webservice aufgerufen, der von der Oracle SOA Suite bereitgestellt wird. Dieser initiale Webservice nimmt nur die Information entgegen, dass sich der Kunde geändert hat und liefert als Antwort ein „OK" zurück. Dies hat den Vorteil, dass für die Oracle Sales Cloud damit die Transaktion abgeschlossen ist und keine Probleme mit der maximalen Groovy Skript-Dauer von einer Minute zu erwarten sind.

Im nächsten Schritt, führt die Oracle SOA Suite über die Standard Webservices der Oracle Sales Cloud ein *„fetch"* aus und fragt mit Hilfe der übergebenen Kunden-ID alle, für das SAP notwendigen, Informationen ab. Diese Daten werden anschließend über einen von SAP bereitgestellten Webservice an die SAP Applikation übergeben. Um den Rückweg zu ermöglichen, wird im gleichen Aufruf die

von SAP zurückgelieferte Kundennummer in die Oracle Sales Cloud geschrieben. Hier muss darauf geachtet werden, dass das Zurückschreiben nicht einen erneuten Call seitens SAP auslöst.

Das Custom Object wurde so aufgebaut, dass es folgende Informationen enthält:

- Welches CRM Objekt wurde übertragen?
- Welche Aktion wurde durchgeführt (Update/Create)?
- Wann wurde die Aktion durchgeführt?
- Wie war die Antwort von der Oracle SOA Suite?
- Konnte die Aktion ebenfalls in der SAP Applikation durchgeführt werden?

Dies hat den Vorteil, dass jedem User ein Integration Log an jedem CRM Objekt zur Verfügung gestellt werden kann. Dies ermöglicht eine Überprüfung der Übertragung zwischen Oracle Sales Cloud und SAP.

In Hinblick auf die Releasefähigkeit der Oracle Sales Cloud ergeben sich durch diesen Ansatz keine Einschränkungen. Darüber hinaus ermöglicht das Nutzen des Custom Objects auf Oracle Sales Cloud-Seite Verbindungsabbrüche zu kompensieren.

Aufbau eines Genehmigung-Workflows

Aufgrund der Verschiebung der Datenhoheit von SAP in die Oracle Sales Cloud ist es nun für den Vertriebsmitarbeiter der Burger GmbH möglich, finanzrelevante Felder in der Oracle Sales Cloud zu verändern. Dies soll nicht möglich sein, sondern muss durch die Finanzabteilung in der Oracle Sales Cloud genehmigt werden. Um dies zu ermöglichen, wurden einige Felder als genehmigungsrelevant definiert. Für diese wurden zuerst drei neue Custom Fields angelegt und wie folgt benannt: *[Feldname]Changed*, *[Feldname]Control*, *[Feldname]Notification*.

Die Felder mit dem Postfix *Changed* werden auf der Edit-Page innerhalb der Oracle Sales Cloud angezeigt und beinhalten den letzten Stand des Wertes. Das ursprüngliche Feld wird nicht auf den Änderungsmasken angezeigt und enthält immer den letzten genehmigten Wert. Dieser wird für den Fall benötigt, dass die Änderung abgelehnt wird und der alte Wert wieder zurückgeschrieben werden soll. Wenn der neue Wert aus dem Changed-Feld akzeptiert wird, so wird dieser in das ursprüngliche Feld geschrieben.

Das Feld mit dem Postfix *Control* enthält drei unterschiedliche Werte, entweder die Änderung wartet auf Genehmigung, die Änderung wurde abgelehnt oder die Änderung wurde genehmigt. Mit Hilfe dieses Felds lässt sich steuern, ob das Feld erneut bearbeitet werden darf oder nicht.

Das Feld mit dem Postfix *Notification* enthält einen Informationstext, damit dem User auf der Genehmigungsmaske weitere Informationen zur durchgeführten Änderung angezeigt werden können. Ein Beispiel für diesen Informationstext: *Kundenname wurde durch Peter Mustermann geändert*.

Darüber hinaus muss pro CRM Objekt ein neues Feld angelegt werden, das zur Steuerung der Masken genutzt wird. Sobald ein genehmigungsrelevantes Feld geändert wurde, wird dem betreffenden User eine andere Maske angezeigt. Aus diesem Grund wird eine neue Maske erstellt, mit der die Genehmigung der einzelnen Änderungen akzeptiert oder abgelehnt werden können. Auf der Standardmaske werden die Felder je nach Status der Änderung nur lesbar angezeigt.

Mit Hilfe von Triggern und Workflows wird im Hintergrund die Steuerung der einzelnen Felder sowie der Masken bewerkstelligt.

Um flexibel bei den Feldern zu sein, wurden alle genehmigungsrelevanten Felder in einem Lookup zusammengefasst. Somit ist diese Lösung leicht auf neue Felder erweiterbar. Denkbar wäre auch, über einen weiteren Workflow einen Task oder eine E-Mail an die User zu verschicken, die die Änderung genehmigen müssen.

Mittels dieses Ansatzes, wird die Releasefähigkeit der Oracle Sales Cloud gewährleistet, da alle zusätzlichen Felder, Trigger und Masken nicht in die Standardlogik eingreifen.

Mandanten-Konzept in der Oracle Sales Cloud

Die Steuerung der Sichtbarkeit von Objekten in der Oracle Sales Cloud wird im Wesentlichen über das Role-Based-Access-Control (RBAC) Konzept gesteuert.

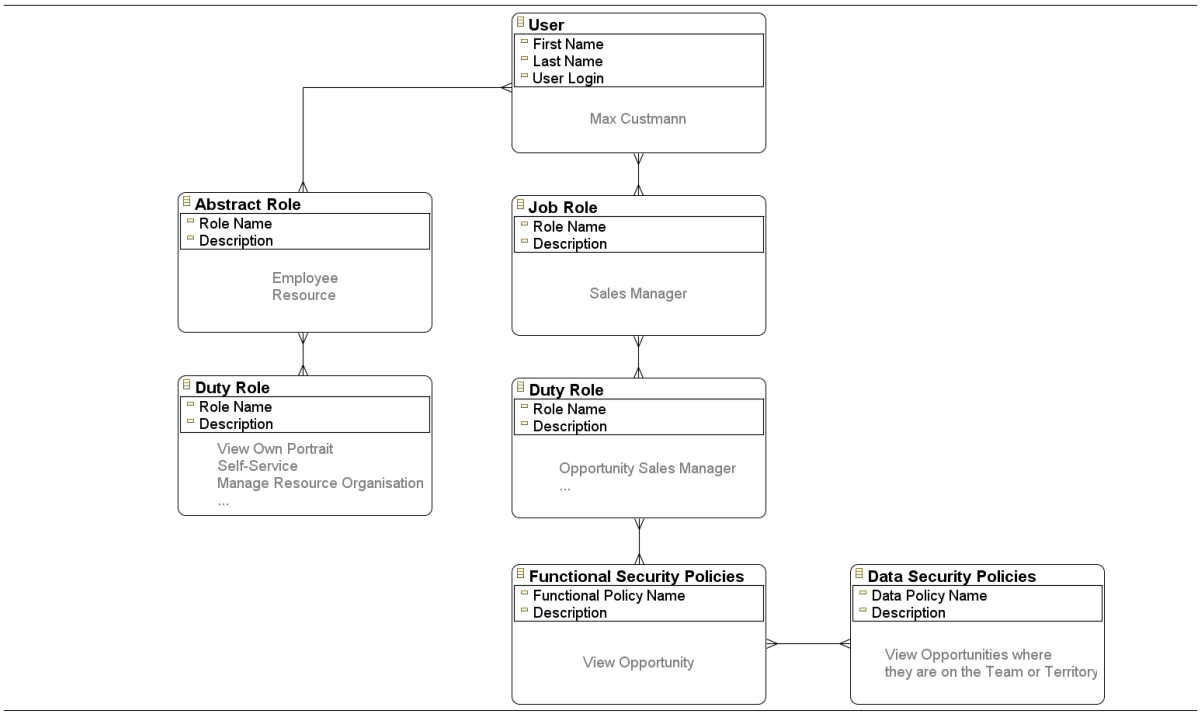

Abbildung 2: Oracle Sales Cloud Role-Based-Access-Control

Die Abbildung 2 veranschaulicht das grundlegende Konzept des Role-Based-Access-Control. Jedem User, der operativ in der Oracle Sales Cloud tätig ist, muss eine Job Role hinzugefügt werden. Jeder Job Role ist eine Vielzahl an Duty Roles zugeordnet. Innerhalb dieser Duty Roles gibt es Functional Security Policies, die determinieren, welche Transaktionen (wie z.B. Erstellen, Löschen oder Lesen von Opportunities) ausgeführt werden dürfen. Diese Regeln sind mit den Data Security Policies verknüpft, welche die Sichtbarkeit der Stamm- oder Bewegungsdaten steuern. Hier kann beispielsweise festgelegt werden, ob ein Sales User „Opportunities“ nur sehen darf, die in einem dem Sales User zugeordneten Vertriebsgebiet sind.

In der Oracle Sales Cloud wurde bis zum Release 10 die Sichtbarkeit der Datensätze primär durch die Mitarbeiterhierarchie, den Vertriebsgebieten sowie der Teamzugehörigkeit gesteuert. Bis zum Release 10 gab es kein Konzept, dass im Standard ermöglichte, Stamm- und Bewegungsdaten wie beispielsweise Leads, Opportunities, Kunden, Kontakte etc. zwischen Mandanten abzugrenzen.

Seit dem Release 11 gibt es out-of-the-box nur die Möglichkeit Leads und Opportunities zwischen Mandanten abzugrenzen. Dies wird ermöglicht, indem ein User an einer Ressourcen-Organisation hängt, die das zusätzliche Attribut „Business Unit“ enthält. Darüber hinaus gibt es nun das Attribut Business Unit an den Objekten „Lead und Opportunity“. Diese Auskunft in Kombination mit der Business Unit-Information am User, ermöglicht das Hinzufügen einer Regel in der Sicherheitskonsole, die den Zugriff für die Objekte Lead und Opportunity ausschließlich gewährleistet, sofern die Business Unit-Informationen übereinstimmen.

Es gibt im Datenmodell der Oracle Sales Cloud für Trading Community Architecture (TCA) für Objekte wie beispielsweise Kunden oder Kontakte kein Business Unit-Attribut. Dies gewährleistet bei einer Standardimplementierung, dass User lesenden Zugriff auf alle Kunden und Kontakte besitzen, unabhängig davon, welche Business Unit in der jeweiligen Ressourcen-Organisation dem User hinterlegt ist.

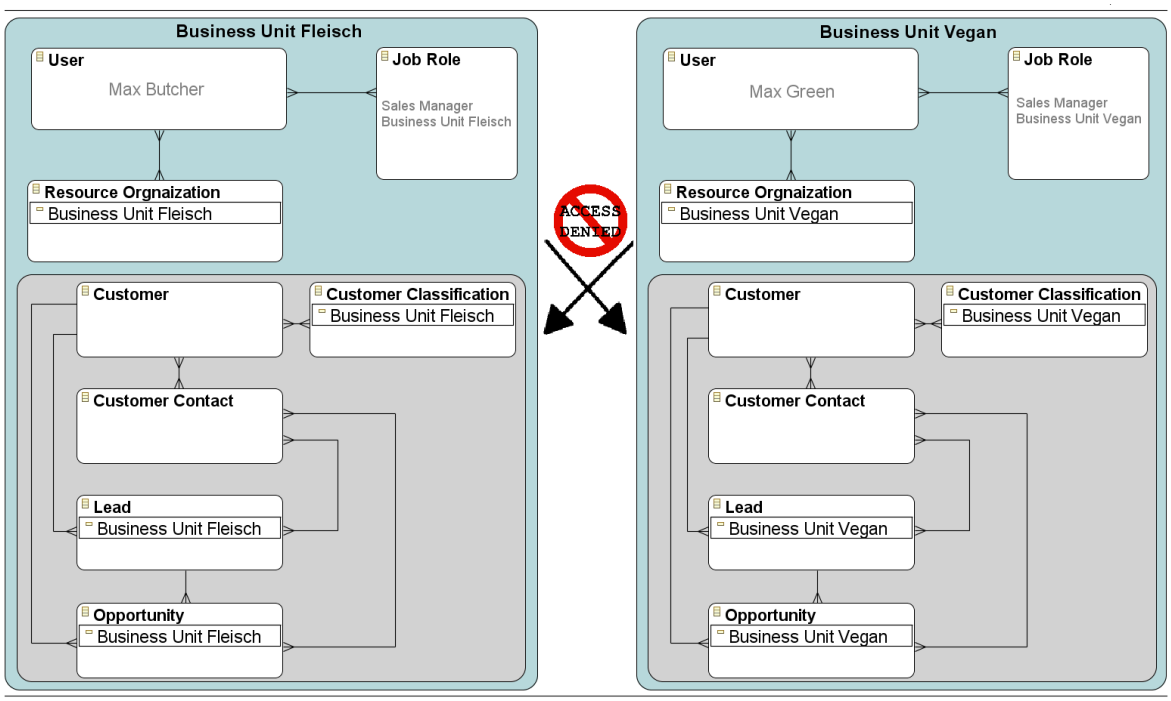

Abbildung 3: Steuerung Sichtbarkeit über Business Units

Abbildung 3 zeigt auf, wie Kunden und Kontakte zwischen Business Units in der Sichtbarkeit abgegrenzt werden können. Um dies zu ermöglichen und so nah wie möglich am Standard zu bleiben, müssen „Customer Classifications“ genutzt werden, die Kunden auf beliebige Art und Weise klassifizieren können. Im Rahmen des Mandanten-Konzeptes können die benötigten Business Units als Klassifikation fungieren. Nachdem jeder Kundendatensatz im Rahmen der initialen Daten-migration eine Business Unit-Klassifikation besitzt, muss jede mandantenspezifische Job Role angepasst werden. Zunächst müssen unter anderem alle Standardregeln deaktiviert werden, die einen allgemeinen Zugriff auf TCA Objekte erlauben. Im Anschluss müssen zwei SQLs (die von Oracle auf Anfrage zur Verfügung gestellt werden) als Data Securtity Policies hinzugefügt werden. Das erste

SQL fragt die Business Unit des eingeloggten Users ab und gleicht diese Information mit der Business Unit ab, die in der Kundenklassifikation hinterlegt ist. Dies ermöglicht bei Übereinstimmung den Zugriff auf einen Kunden. Das zweite SQL, welches den Zugriff auf Kontakte steuert, prüft zuerst die Verbindung des Kontakts zum Kunden und gleicht anschließend analog zum ersten SQL die Business Unit-Informationen zwischen Kunden und User ab. Dieser Schritt ist essentiell, denn es gibt am Kontakt keine Klassifikation, so dass die Abgrenzung nur für Kontakte funktioniert, die eine Beziehung zu einem Kunden haben. Vor dem Hintergrund, dass Kontakte in einem CRM System ohne Bezug zu einem Kunden selten vorzufinden sind, ist dieser Nachteil zu vernachlässigen. Sollte es ein Szenario geben, wo es viele losgelöste Kontakte gibt, sind auch in der Oracle Sales Cloud stabile Lösungsansätze möglich, die hier nicht weiter betrachtet werden.

Die Akzeptanz dieses Ansatzes steht und fällt mit dem automatischen Setzen der Klassifikation am Kunden. Damit der End-User nicht jeden Kunden selber klassifizieren muss, kann im Application Composer ein Groovy Script geschrieben werden, welches in Abhängigkeit der Job Role oder der Business Unit des eingeloggten Users die Klassifikation automatisch setzt.

In Hinblick auf Wartung und Releasefähigkeit der mandantenspezifischen Abgrenzung von TCA Objekten sollten immer die Klassifikationen genutzt werden. Obwohl auch ein Custom Attribut (z.B. Business Unit am Kunden) in einem eigen entwickelten SQL die Grundlage für die Abgrenzung bilden kann, sollte in Bezug auf die oben genannte Wartung und Releasefähigkeit hiervon abgesehen werden.

Dies gilt nicht für Custom Objects; hier können ohne Bedenken definierte Attribute als Abgrenzung dienen.

Audit Log

Neben dem Genehmigungs-Workflow sollen auch alle anderen Änderungen an Feldern von CRM Objekten aufgezeichnet werden. Hierzu sollte eine allgemeingültige und erweiterbare Möglichkeit geschaffen werden, die das Aufzeichnen von Aktualisierungen auf definierten Feldern ermöglicht. Um die Erweiterbarkeit zu gewährleisten, wurde ein Lookup pro Objekt eingerichtet. Dieser Lookup enthält die API-Namen der Felder, die im Log aufgezeichnet werden sollen. Somit ist es für den Sales Cloud-Administrator der Burger GmbH einfach, die Felder zu erweitern oder Felder wieder zu entfernen.

Über einen *Before Upate Trigger* wird eine Groovy Funktion aufgerufen. Diese Groovy Funktion liest zuerst den Lookup aus und prüft für alle Felder über die Standard Groovy Funktion *isAttributeChanged()*, ob der Wert des Felds geändert wurde. Liefert diese Funktion *true* zurück, wird mit einem Zeitstempel den Namen des Felds und des neuen Wertes der Logeintrag zusammengebaut und in ein Textfeld geschrieben. Dieses Feld wird auf einer Detailmaske an den jeweiligen CRM Objekten angezeigt.

Im Folgenden ein Groovy Code-Ausschnitt aus der Funktion:

```
def vo = newView('CommonLookup')
def vc = newViewCriteria(vo)
def vcr = vc.createRow()
def vci = vcr.ensureCriteriaItem('LookupType')
vci.setOperator('=')
vci.setValue('XXPROM_FIELD_LOG')
vc.insertRow(vcr)
vo.appendViewCriteria(vc)
vo.executeQuery()
while (vo.hasNext())
{
  def curRow = vo.next()
  newval = ""
  try
  {
    if(isAttributeChanged(curRow.Meaning))
    {
      log = "" + Now +" - " +displayName + " changed field: " + curRow.Description +" to "+ getAttribute (curRow.Meaning)
+"\n" + log;
    }
  }
}
```

Kontaktadresse:

Wassili Billon		Yves Chassein	
PROMATIS software GmbH		PROMATIS software GmbH	
Notkestr. 9		Pforzheimer Straße 160	
D-22607 Hamburg		D-76275 Ettlingen	
Telefon:	+49 (0) 40 2533269 0	Telefon:	+49 (0) 7243 2179 0
Fax:	+49 (0) 7243 2179 999	Fax:	+49 (0) 7243 2179 999
E-Mail	wassili.billon@promatis.de	E-Mail	yves.chassein@promatis.de
Internet:	www.promatis.de	Internet:	www.promatis.de