Funktionsbeschreibung der OSCI-Bibliothek (JAVA)

Version 1.04

19.10.2011

Herausgeberin: Freie Hansestadt Bremen

Vertrieb: Projektbüro KoopA ADV

Erstellung: bremen online services GmbH & Co. KG

Inhaltsverzeichnis

1 Übersicht und Zielsetzung

Die Aufgabe der im Folgenden beschriebenen OSCI-Bibliothek besteht darin, Anwendungsprogrammen und Fachverfahren eine Softwarekomponente zur Verfügung zu stellen, mit deren Hilfe sie das OSCI-Transportprotokoll zum Erzeugen und Empfangen von Nachrichten gemäß der OSCI-Transport-1.2-Spezifikation unter Berücksichtigung der Änderungen aus der Korrigenda vom 10.06.2004 nutzen können. Die Bibliothek soll eine möglichst einfache Integration von OSCI 1.2 in bestehende Systeme ermöglichen. Im Zusammenwirken mit den später noch näher zu erläuternden externen Modulen umfasst sie alle für Benutzer im Sinne der Spezifikation erforderlichen Funktionalitäten, nämlich Aufbau, Versand, Empfang, Speicherung, Ver- und Entschlüsselung sowie die mathematische Signaturprüfung sämtlicher Nachrichtentypen. Mit Blick auf die Fachverfahrensintegration bildet die Schnittstelle der OSCI-Bibliothek den Zugang zu einer kompletten OSCI-Infrastruktur und ist somit der einzige Punkt, an dem eine Anpassung an die unterschiedlichen Fachverfahren erfolgen muss.

Das vorliegende Dokument setzt Kenntnisse der OSCI-Spezifikation 1.2 sowie der Begleitdokumentation zur OSCI-Spezifikation „Entwurfsprinzipien, Sicherheitsziele und -mechanismen" voraus. Die in der OSCI-1.2-Spezifikation erwähnten Literaturhinweise gelten ebenfalls für dieses Dokument. Besonders hervorzuheben sind hier die SOAP-Spezifikation, die Spezifikation für SOAP mit Attachment, die XML-Signature-Spezifikation sowie die XML-Encryption-Spezifikation.

Weitere wichtige Voraussetzungen zum Verständnis des vorliegenden Dokuments sind Kenntnisse in der Programmiersprache JAVA™ sowie ein Überblick über die von SUN® zur Verfügung gestellte Kryptografieschnittstelle JCA/JCE. Die vorliegende Dokumentation bezieht sich auf die Implementierung in der Programmiersprache JAVA.

Im folgenden Kapitel wird der Leistungsumfang der OSCI-Bibliothek geschildert. Nach der Interpretation der OSCI-Spezifikation folgen eine Beschreibung der Schnittstellen zu externen Modulen, Beschreibungen der Programmierschnittstelle, der Fehlerbehandlung sowie des grundsätzlichen Aufbaus der Bibliothek. Abschließend werden anhand mehrerer Use-Cases die Abläufe für verschiedene Nachrichtentypen dargestellt.

2 Beschreibung des Leistungsumfangs (zugesicherte Eigenschaften)

Die zentrale Aufgabe der Bibliothek besteht in der Komposition valider OSCI-Nachrichten für den Versand, sowie der Dekomposition von OSCI-Nachrichten bei deren Empfang und der Prüfung ihrer syntaktischen Korrektheit. Weiter werden alle kryptografischen Funktionen zur Signaturerzeugung und -prüfung sowie Ver- und Entschlüsselung über Funktionen der Bibliothek verwaltet, wobei die eigentliche Abarbeitung kryptografischer Aufgaben inkl. der Zugriffe auf Crypto-Token durch entsprechende Drittimplementierungen zur Verfügung gestellt werden müssen.

In diesem Sinne werden durch die Bibliothek alle für den vollständigen Aufbau einer OSCI-Nachricht benötigten Objekte wie die eigentlichen Inhaltsdaten, die Signatur- und Verschlüsselungszertifikate aller Kommunikationsbeteiligten und deren physikalische Adressen in Form entsprechender Klassen bzw. derer Attribute bereitgestellt.

Die Erstellung und Weiterverarbeitung von Inhaltsdaten ist nicht Aufgabe der Bibliothek, sondern muss durch die Client-Systeme und Fachanwendungen erfolgen. Durch die Klassen und Methoden der Bibliothek wird lediglich sichergestellt, dass diese Schema-konform in die Nachrichtenobjekte eingebettet werden.

Eine weitere wesentliche Aufgabe der Bibliothek besteht in der Steuerung und Überwachung des Kommunikationsvorgangs. Für diesen Zweck stellt die Bibliothek Klassen und Methoden zur Verfügung, anhand derer die Plausibilität einer Kommunikation überprüft und dokumentiert werden kann.

Methoden für den technischen Versand und Empfang von OSCI-Nachrichten (z.B. auf Basis des HTTP-Protokolls), die Bereitstellung kryptografischer Funktionen für die Signatur und Verschlüsselung von Nachrichten sowie die Visualisierung signierter Nachrichtenkomponenten sind nicht Bestandteil der Bibliothek. Diese Funktionalitäten müssen durch separate Module realisiert und durch JAVA-Interfaces der Bibliothek eingebunden werden. Dies erlaubt den Einsatz unterschiedlicher Module für den jeweiligen Aufgabenbereich.

Außerhalb des Funktionsumfangs und der Kontrolle der Bibliothek liegt die *TrustViewerService*-Implementierung zur sicheren Visualisierung der zu signierenden Inhaltsdaten.

OSCI Transport 1.2 lässt beliebig viele Inhaltsdatencontainer zu. Die Bibliothek stellt die Funktionalitäten zur Verfügung, mit denen diese Inhaltsdatencontainer konform zur OSCI-Spezifikation aufgebaut werden können.

Die vorliegende Implementierung wurde für eine JAVA-Laufzeitumgebung der Version 1.5 entwickelt. Die Bibliothek selbst greift nicht auf plattformspezifischen (native) Code zu und ist daher auf allen Systemen einsetzbar, für die eine solche Laufzeitumgebung verfügbar ist.

Bei dem Design der OSCI-Bibliothek wurde versucht, größtmöglichen Komfort für den Benutzer zu erreichen ohne ihn in seinen Möglichkeiten einzuschränken. Die Grenzen der OSCI-Bibliothek sollen im nächsten Schaubild dargestellt werden.

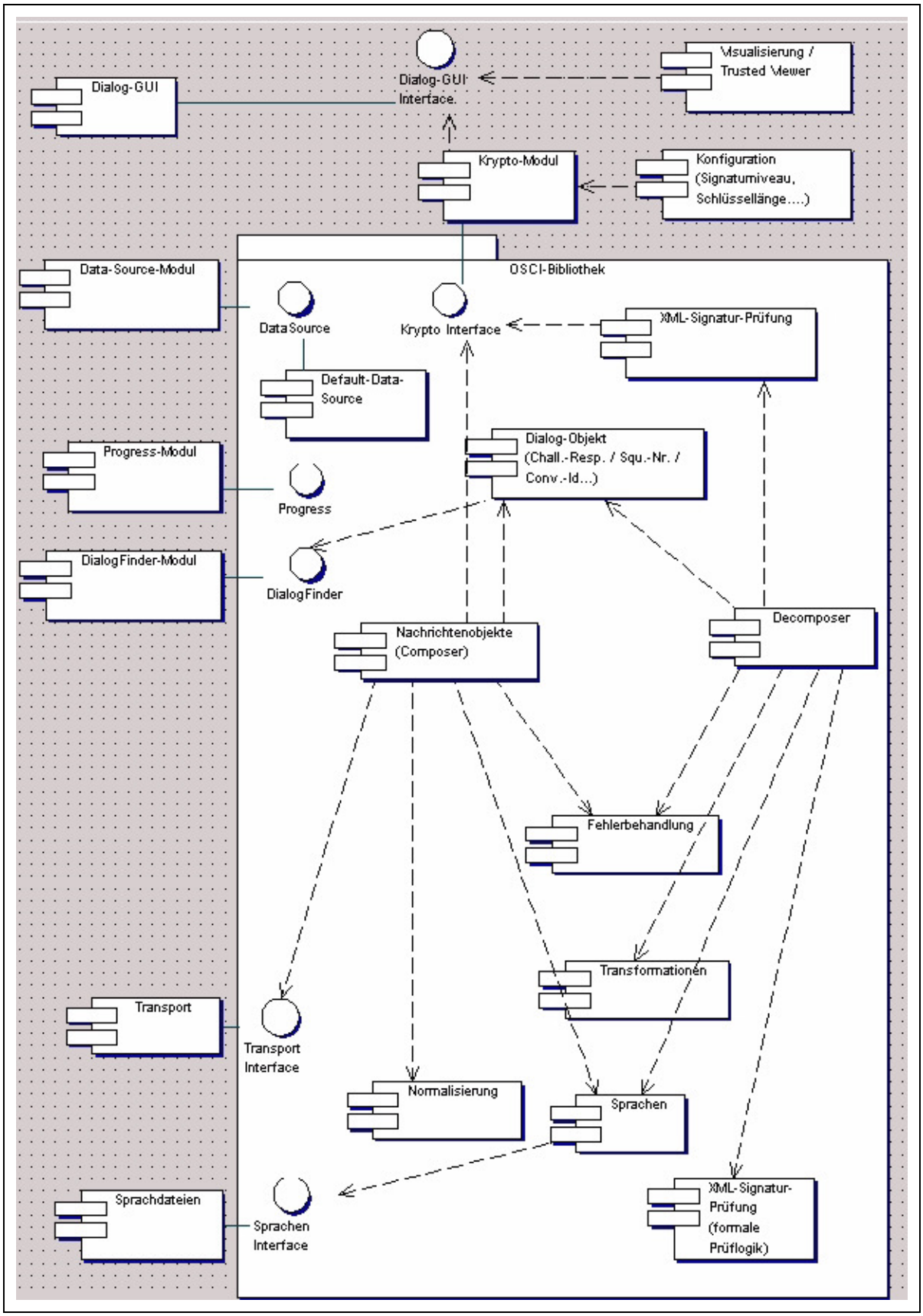

Abbildung 1: Überblick über die OSCI-Bibliothek

Weitere Details zu den Grenzen der OSCI-Bibliothek werden im Kapitel 4 beschrieben.

3 Interpretation der OSCI-Spezifikation

Die OSCI-Spezifikation definiert XML-Schemata, mit denen die Struktur der übertragenen Nachrichten festgelegt wird. Die OSCI-Spezifikation schreibt die Struktur der verschlüsselten und/oder signierten Inhaltsdaten vor. Die folgende Abbildung stellt die Vorgaben der OSCI-Spezifikation dar.

Die Spezifikation macht in einigen Punkten keine konkreten, detaillierten Angaben darüber, wie die Inhaltsdaten mit diesen Schemata behandelt werden sollen. Dies betrifft u.a. die Behandlung von verschlüsselten Daten und Attachments. Hieraus ergibt sich hinsichtlich der Verwendung der Schemata ein gewisser Interpretationsspielraum, für den in Absprache mit der OSCI-Leitstelle bei der Realisierung der OSCI-Bibliothek folgende Einschränkungen bzw. Konkretisierungen vorgenommen wurden:

- Unterhalb eines ContentPackage-Elementes befinden sich lediglich ContentContainer oder verschlüsselte ContentContainer-Elemente, die in EncryptedData-Elementen enthalten sind.
- Inhaltsdaten werden grundsätzlich in Content-Elementen untergebracht, die wiederum Unterelemente von ContentContainer-Elementen sind. Content-Elemente können statt Inhaltsdaten auch ein href-Attribut enthalten, welches die Referenz auf ein der Nachricht hinzugefügtes Attachment enthält. Ein solcher Eintrag hat z.B. folgende Form:

  ```
  <osci:Content Id="content1" href="cid:d:/xplog.txt"/>
  ```

  Alle Attachments einer Nachricht müssen über einen solchen Eintrag referenziert werden. Die Bibliothek stellt für Content- und ContentContainer-Elemente gleichnamige Klassen zur Verfügung, mit denen entsprechende Objekte angelegt werden können. Die ContentContainer-Klasse bietet eine Methode für das Hinzufügen von Content-Objekten.
- EncryptedData-Elemente enthalten grundsätzlich nur verschlüsselte ContentContainer-Elemente. Die Bibliothek enthält eine entsprechende EncryptedData-Klasse, deren Konstruktor ein ContentContainer-Objekt übergeben werden kann. Enthält ein zu verschlüsselnder ContentContainer ein Content-Element mit einer Attachmentreferenz, so wird dem ContentContainer beim Verschlüsseln automatisch ein EncryptedData-Element hinzugefügt, welches die Verschlüsselungsinformationen für das Attachment enthält.
- Die Signaturen von Inhaltsdaten werden gemäß OSCI-Schema in den ContentContainer neben den zu signierenden Content- oder EncryptedData-Elementen eingesetzt. Die OSCI-Bibliothek signiert grundsätzlich alle in dem ContentContainer enthaltenen Elemente, außerdem ggf. in Content-Elementen referenzierte Attachments.

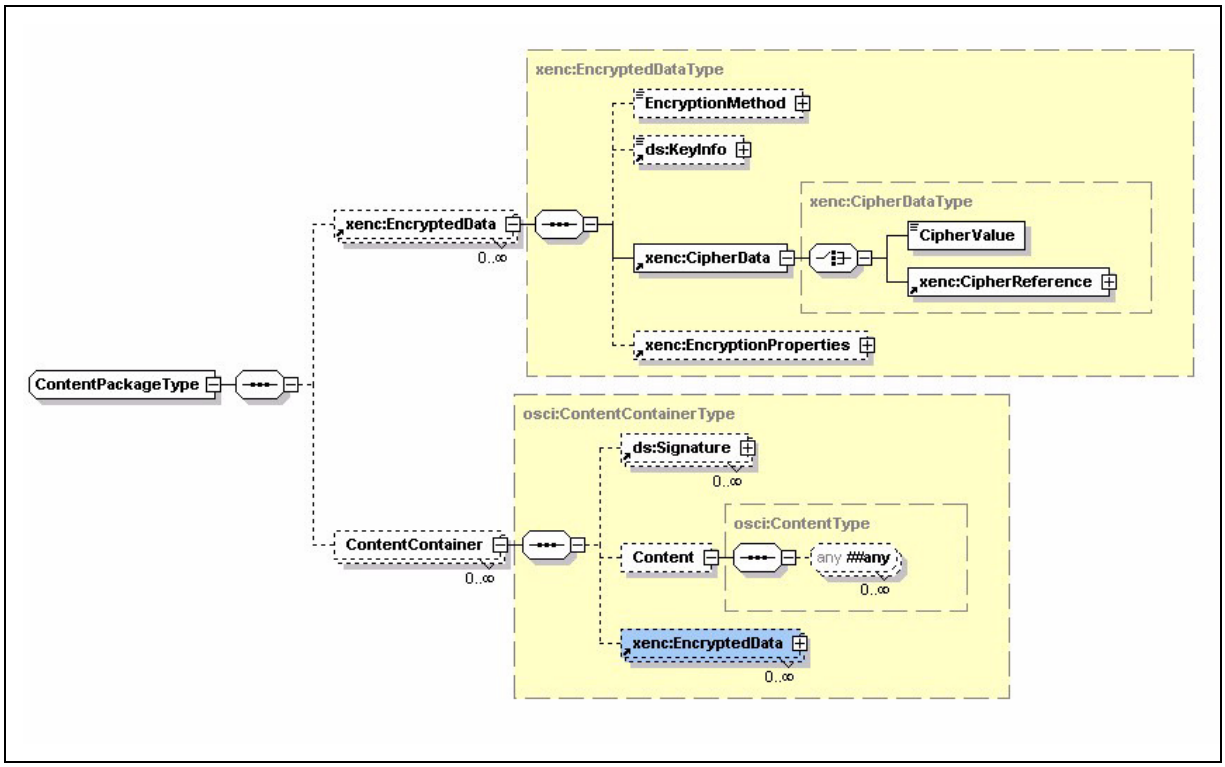

Abbildung 2: Aufbau der Inhaltsdaten laut OSCI-Spezifikation

4 Schnittstellen der Bibliothek zu externen Modulen

Für den Einsatz der OSCI-Bibliothek ist die Kombination mit externen Modulen erforderlich. Die Schnittstellen zu diesen Modulen gewährleisten sowohl Freiheit beim Einsatz neuer Technologien (z.B. bezüglich der krytografischen Funktionen) als auch die Möglichkeit, die OSCI-Bibliothek an die vorhandene Infrastruktur (z.B. Datenbanken) anzubinden.

In der Abbildung 3 sind die Interfaces und Klassen dargestellt.

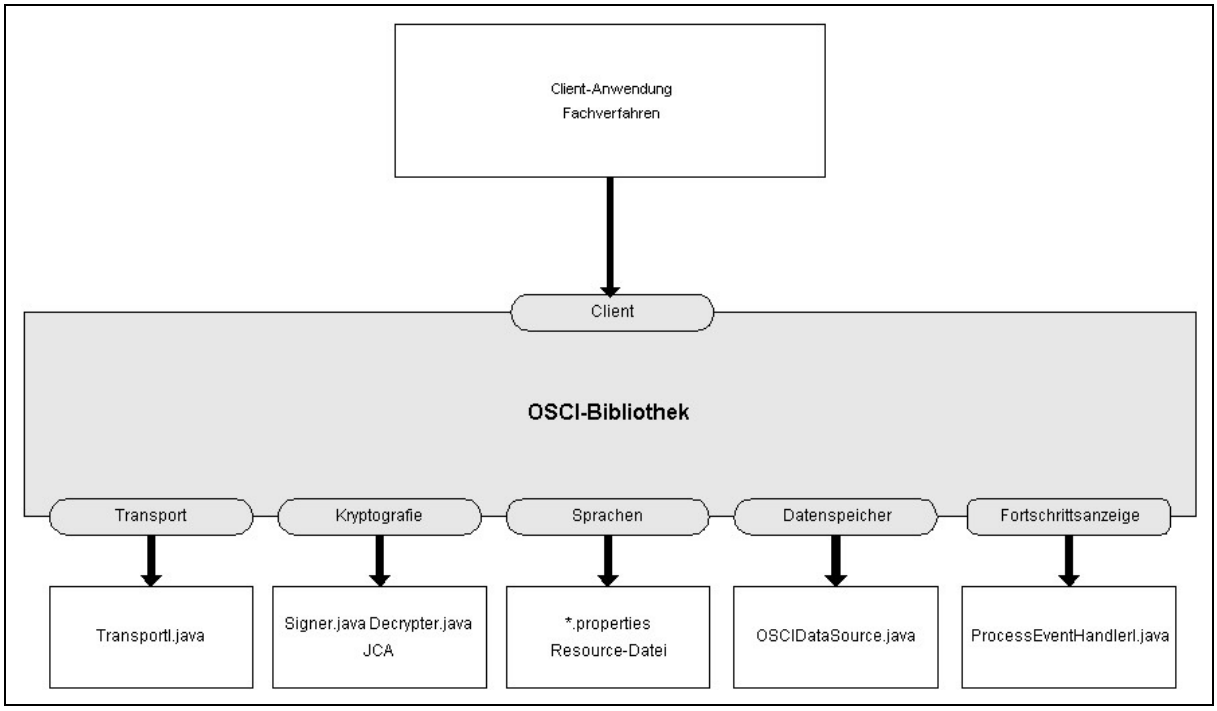

Abbildung 3: Bibliothek mit Interfaces

4.1 Crypto-Interface

Die OSCI-Bibliothek übernimmt den Aufbau und das Parsen der OSCI-Dokumente. Hierzu gehört insbesondere das Anbringen der Signaturen und die Ver- bzw. Entschlüsselung.

Die OSCI-Bibliothek muss zur Erfüllung ihrer Aufgaben folgende Operationen durchführen, für die kryptographische Algorithmen erforderlich sind:

- Erzeugung und Prüfung von Signaturen
- Ver- und Entschlüsselung von Daten
- Erzeugung von Challenge-Werten
- Erzeugung von kryptografischen Schlüsseln bzw. Schlüsselpaaren

Anwender der Bibliothek sollen ihre bereits vorhandenen Security-Module möglichst weiterverwenden können. Neben der Einbindung eigener Implementierungen der benötigten Algorithmen steht die Verwendung unterschiedlichster Schlüssel- und Zertifikatsressourcen im Vordergrund. Darüber hinaus muss die Verwendung von Crypto-Tokens ermöglicht werden, die neben der Bereitstellung von Schlüsseln und Zertifikaten auch die kryptografischen Operationen selbst durchführen (PKCS#11-Module, SmartCards etc.).

4.1.1 Aufgabe der Schnittstelle

Im JAVA-Umfeld hat sich für die Durchführung kryptographischer Funktionen der Einsatz von JCA/JCE bewährt. Die JCA/JCE-Schnittstellen erlauben es den Nutzern, eigene Service-Provider zu schreiben und zu registrieren, so dass kein neues Interface definiert werden musste, sondern auf bewährte Konzepte zurückgegriffen werden konnte.

Im Rahmen der OSCI-Spezifikation sind die nachfolgend aufgezählten Algorithmen vorgeschrieben worden. Außerdem wird eine Methode zur sicheren Erzeugung von Zufallszahlen benötigt.

- Verschlüsselung
  - RSA
  - TDES
  - AES-128/192/256
- Signatur (Hashen)
  - RIPEMD-160
  - SHA-1

4.1.2 Aufbau der Schnittstelle

Die JAVA-Sicherheitsarchitektur stellt mit der JCA/JCE-Architektur bereits Schnittstellen, sogenannte Service Provider Interfaces zur Verfügung, die die Anbindung unterschiedlicher Implementierungen ermöglichen. Für die Bibliothek sind dabei diese Interfaces von Bedeutung:

- `javax.crypto.CipherSpi`
- `java.security.MessageDigestSpi`
- `java.security.SecureRandomSpi`
- `javax.crypto.KeyGeneratorSpi`

Für die Anbindung von Crypto-Tokens wurden zwei abstrakte Klassen für die sicherheitsrelevanten Funktionen Entschlüsseln und Signieren vorgesehen:

- `de.osci.osci12.common.Decrypter`
- `de.osci.osci12.common.Signer`

Anwender müssen Implementierungen dieser Klassen an die Rollenobjekte für eine OSCI-Nachricht bzw. einen OSCI-Dialog übergeben.

4.1.2.1 Signer

Eine Implementierung der Signer-Klasse muss zwei Methoden enthalten, `getCertificate()` und `sign(byte[] hash, String algorithm)`. Erstere muss das Signaturzertifikat des signierenden Rollenobjekts zurückgeben, die zweite die Signatur eines Byte-Arrays durchführen und das Ergebnis ebenfalls als Byte-Array zurückgeben.

4.1.2.2 Decrypter

Eine Implementierung der Decrypter-Klasse muss ebenfalls zwei Methoden enthalten, nämlich `getCertificate()` und `decrypt(byte[] data)`. Erstere muss das Verschlüsselungszertifikat des verschlüsselnden Rollenobjekts zurückgeben, die zweite die Entschlüsselung eines Byte-Arrays durchführen und das Ergebnis ebenfalls als Byte-Array zurückgeben.

4.2 Transport-Schnittstelle

Das Transport-Interface ist für die Übermittlung einer fertiggestellten OSCI-Nachricht an den Empfänger vorgesehen. Das Transport-Interface ist daher unentbehrlich für die OSCI-Bibliothek und muss auf jeden Fall implementiert werden.

4.2.1 Aufgaben der Schnittstelle

Die OSCI-Bibliothek stellt die Funktionalität zum Aufbau einer vollständigen OSCI-Nachricht bereit, nicht jedoch zu deren Übermittlung.

Je nach konkretem Einsatzszenario können für die Übermittlung der Nachricht unterschiedliche Protokolle zum Einsatz kommen; möglich sind z.B. http, https, ftp, smtp/pop oder jms. Alle genannten Protokolle sollten sich auf Grundlage des bereitgestellten Interfaces umsetzen lassen. Dies führte zu der Entscheidung, keinerlei protokollspezifische Eigenheiten in der Bibliothek zu berücksichtigen.

Im Prinzip hätte für den Transport der Nachricht kein Interface definiert werden müssen. Die Alternative wäre gewesen, dass die OSCI-Bibliothek eine Methode für das Auslesen der fertig aufgebauten Nachricht bereitstellt und den Transport damit vollständig in die Anwendung verlagert. Eine empfangene Nachricht würde dann einfach von der Anwendung wieder in die Bibliothek eingestellt.

Die Entscheidung, an dieser Stelle dennoch ein Interface zu definieren und den Transport somit enger mit der Bibliothek zu verzahnen, wurde primär mit dem Ziel der einfachen Austauschbarkeit der Transportkomponente getroffen.

Die Verwendung eines Interface bewirkt, dass die Bibliothek die Kontrolle erst wieder an die Applikation abgibt, nachdem die Antwortnachricht komplett aufgebaut ist. Dadurch kann die Bibliothek sogleich die Korrektheit der Dialogparameter überprüfen und eventuell einen Lesevorgang abbrechen.

4.2.2 Aufbau der Schnittstelle

Die zu verwendende Implementierung des Transport-Interfaces wird der Bibliothek durch den Konstruktor des DialogHandlers bekannt gegeben und kann während eines Dialoges nicht gewechselt werden.

Implementierungen des Interface haben lediglich die Aufgabe, den reinen Transport der Nachricht vom Sender zum Empfänger zu realisieren. Insbesondere muss sich diese Schicht nicht um den Aufbau der SOAP-Strukturen kümmern, da diese Bestandteil der OSCI-Spezifikation sind. Das bedeutet auch, dass Fehler auf SOAP-Ebene auf beiden Seiten von der Bibliothek behandelt werden. Eine solche Fehlernachricht ist für die Transportschicht damit eine Nachricht wie jede andere.

Da die Art und Weise, wie die Zieladresse einer Nachricht aufgebaut ist, stark vom Protokoll abhängt, wird diese sehr allgemein als URI formuliert. Eine URI kann in JAVA einfach in eine URL umgewandelt werden.

Nicht zum Interface gehört z.B. eine Methode für das Setzen eines Timeout-Werts. Dieser Wert muss abhängig vom Anwendungsszenario oder sogar in Abhängigkeit der konkreten Nachricht(engröße) festgelegt werden und gehört daher zur Anwendungslogik.

Je nach Verbindungstyp muss sich der User unter Umständen authentifizieren. Auch diese Aufgaben wurden in die Transportschicht ausgelagert, da sie protokollspezifisch sind.

4.2.3 Methoden des TransportI-Interface

Package: `de.osci.osci12.extinterfaces.transport`

- `public String getVersion()`

  Liefert die Versionsnummer.

- `public String getVendor()`

  Gibt den Namen des Herstellers zurück.

- `public InputStream getResponseStream()`

  Liefert den Response Stream.

- `public boolean isOnline(URI uri)`

  Mit dieser Methode soll die Anwendung die Erreichbarkeit einer URL testen können. Die Methode wird von der Bibliothek nicht aufgerufen und kann daher auch leer implementiert werden (z.B. `return false;`).

- `public long getContentLength()`

  Diese Methode soll die Länge des Response-Streams der Antwortnachricht zurückgeben, sofern das verwendete Transportprotokoll diese Information (beim Beginn des Lesevorgangs) liefert. Auch diese Methode wird zur Zeit noch nicht von der Bibliothek verwendet.

- `public OutputStream getConnection(URI uri, long length)`

  Liefert eine konkrete Verbindung zum Versenden einer Nachricht. Die Methode konnektet zu der übergebenen URI und liefert als Ergebnis einen Output-Stream in den die Bibliothek dann die OSCI-Nachricht serialisiert. Der Parameter wird zur Zeit nicht unterstützt und von der Bibliothek als –1 übergeben.

Eine detailliertere Beschreibung dieser Methoden findet sich in der JavaDoc-Dokumentation.

4.2.4 Ablauf beim Senden einer Nachricht

Folgende Schritte werden von der Bibliothek beim Versenden ausgeführt:

- Über die Methode `getConnection(...)` holt sich die Bibliothek von der Transport-Implementierung einen Output-Stream.
- Die Bibliothek schreibt die OSCI-Nachricht in den Output-Stream und schließt den Stream (`close()`).
- Nach dem Versenden holt sich die Bibliothek den Response-Input-Stream zur Nachricht durch die Methode `(getResponseStream()).`

4.3 Progress-Schnittstelle

Die Progress-Schnittstelle wird von der Bibliothek für die Anzeige von Statusmeldungen benutzt. Diese Schnittstelle wird im Zusammenhang mit zeitaufwendigen Aktionen benutzt, z.B. bei der Übermittlung von Nachrichten über die Transportschicht, beim Ver- und Entschlüsseln sowie beim Parsen von OSCI-Nachrichten. Auch bei Sequenzen von Abläufen kann die Bibliothek durch die Progress-Schnittstelle mitteilen, welcher Bearbeitungsschritt gerade durchgeführt wird.

4.3.1 Aufgaben der Schnittstelle

Die OSCI-Bibliothek kommuniziert über die Progress-Schnittstelle ihre Statusmeldungen an die Anwendung. Da die Bibliothek kein eigenes GUI besitzt, muss für die Anzeige von Statusinformationen der Bibliothek ein Interface implementiert werden.

Die grafische Darstellung von Fortschritt-Statusereignissen ist ebenfalls applikationsabhängig. Auch in diesem Fall ist daher ein Interface notwendig, welches der Applikation genau diese Informationen übermittelt. Die Progress-Schnittstelle wird benutzt, um den Arbeitsfortschritt einer Tätigkeit in Prozentschritten anzuzeigen oder um mitzuteilen, welche Tätigkeit gerade von der Bibliothek ausgeführt wird.

4.3.2 Aufbau der Schnittstelle

Die Schnittstelle liegt in Form eines JAVA-Interface vor. Die Registriertung der Progress-Implementierung erfolgt durch die Methode `setProgressEventHandler(ProgressEventHandlerI progressEventHandler)` im Dialog-Handler. Die Registrierung einer Implementierung ist optional.

4.3.3 Methoden der Progress-Schnittstelle

Package: `de.osci.osci12.extinterfaces.progress`

- `public String getVersion()`

  Liefert die Versionsnummer.

- `public String getVendor()`

  Sollte den Namen des Herstellers zurückgeben.

- `public void event(int type, String param, int percent)`

  Wird aufgerufen, sobald zeitaufwendige Vorgänge in der Bibliothek der Applikation mitgeteilt werden sollen. Die Identifier der Events (type) werden in der Klasse `de.osci.osci12.common.Constants` definiert und bei Bedarf ergänzt.

Eine detailliertere Beschreibung dieser Methoden findet sich in der JavaDoc-Dokumentation.

4.3.4 Ablauf

Für die OSCI-Bibliothek ist ausschließlich die Methode `event(int type, String param, int percent)` von Bedeutung. Diese wird von der Bibliothek bei Fortschritt-Statusereignissen aufgerufen.

4.4 Language-Schnittstelle

Die Language-Schnittstelle dient zur Übersetzung der Fehler-, Warnung- und Progresscodes in die jeweilige Landessprache. Die Sprachen Deutsch und Englisch sind schon Bestandteil der OSCI-Bibliothek, weitere Sprachen können bei Bedarf durch die Nutzung von Resource-Dateien hinzugefügt werden.

Die OSCI-Bibliothek liefert zu jeder Meldung einen Code, der durch die Übersetzungstabellen der Resource-Language-Dateien dann sogleich den Meldungstext in der gewünschten Landessprache referenziert.

4.4.1 Aufbau der Schnittstelle

Die Language-Schnittstelle ist kein JAVA-Interface im eigentlichen Sinne, vielmehr stellt sie eine JAVA-Resource-Datei dar. Für jede Sprache ist eine Datei zu erstellen, die in folgendes Verzeichnis einzufügen ist: `de/osci/osci12/extinterfaces/language`.

Der Dateiname besteht aus dem String "Text_", dem angehängten zweistelligen Sprachkürzel nach ISO-639 und der Erweiterung "properties". Die deutsche Sprachdatei heißt somit "Text_de.properties". Sollte die gewünschte Landessprache nicht vorhanden sein, wird auf die englischsprachige Datei zurückgegriffen

4.4.2 Ablauf

Die Bibliothek verwendet die gemäß Ländereinstellung des Systems passende Resource-Datei und kommuniziert alle Anfragen in der entsprechenden Landessprache.

4.4.3 Auszug aus der deutschen Resource-Datei

Die folgenden Meldungstexte stammen aus der deutschen Resource-Datei "Text_de.properties". Mit relativ wenig Aufwand können natürlich auch andere Landessprachen unterstützt werden können.

4.4.3.1 OSCI-Fehlermeldungen auf Nachrichtenebene

9000 = Interner Fehler beim Supplier.

9100 = Empfangene Nachricht stellt keine gültige OSCI-Nachricht dar.

9200 = Supplier verfügt nicht über den privaten Schlüssel zum Chiffrierzertifikat auf Nachrichtenebene.

[...]

4.4.3.2 OSCI-Rückmeldungen auf Auftragsebene

3500 = Chiffrierzertifikat des Clients ist zeitlich ungültig.

3501 = Prüfung des Chiffrierzertifikats des Clients konnte nicht vollständig durchgeführt werden.

3700 = Signierzertifikat des Clients ist zeitlich ungültig.

3701 = Signierzertifikat eines Autors ist zeitlich ungültig.

3702 = Signatur über das Signierzertifikat eines Autors ist fehlerhaft.

[...]

0800 = Auftrag ausgeführt, Dialog beendet.

0801 = Auftrag ausgeführt, Dialog weiterhin geöffnet.

9600 = Auftrag ist nicht signiert, obwohl dieser Supplier für diesen Auftragstyp eine Signatur verlangt.

9601 = Signatur über Auftrag ist fehlerhaft.

[...]

4.4.3.3 Fehlermeldungen aus der Bibliothek

no_cipher_cert_intermed = Kein Verschlüsselungszertifikat für das Rollenobjekt eingestellt.

no_decrypter_intermed = Kein Decrypter-Objekt für den Intermediär eingestellt.

no_signature_cert_intermed = Kein Signaturzertifikat für das Rollenobjekt eingestellt.

no_signer_intermed = Kein Signer-Objekt für den Intermediär eingestellt.

[...]

4.5 Datenspeicher-Schnittstelle (OSCIDataSource)

Die Datenspeicher-Schnittstelle dient zur Anbindung beliebiger Speichersysteme, etwa Datenbanken oder Dateisysteme. Diese Schnittstelle wird von der OSCI-Bibliothek z.B. genutzt, um eingelesene Daten vor der Signatur oder empfangene Daten vor der Entschlüsselung temporär zwischenzuspeichern.

4.5.1 Aufgaben der Schnittstelle

Nachrichten können - von Ausnahmen wie z.B. InitDialog-Nachrichten abgesehen - in der Regel nicht in einem zeitlich zusammenhängenden Vorgang verarbeitet werden. Beispielsweise findet die Ver- oder Entschlüsselung von Inhaltsdaten als Teil einer Nachricht häufig später als der unmittelbare Versand oder Empfang der Nachricht statt. Ähnliches gilt für die Erzeugung oder Prüfung von

Signaturen. Aus diesem Grund muss die Bibliothek an verschiedenen Stellen des Verarbeitungsprozesses temporäre Kopien von Nachrichten (oder Teilen davon) anlegen. Die DataSource-Schnittstelle wurde definiert, um den Betrieb der OSCI-Bibliothek in unterschiedlichen Umgebungen zu gewährleisten.

4.5.2 Aufbau der Schnittstelle

Die Schnittstelle liegt in Form der abstrakten Klasse `de.osci.osci12.common.OSCIDataSource` vor. Eine Implementierung dieser Klasse kann in der Klasse `de.osci.osci12.common.DialogHandler` mit Hilfe der statischen Methode `setDataBuffer(OSCIDataSource buffer)` für die Verwendung durch die Bibliothek installiert werden. Die Implementierung `de.osci.osci12.common.SwapBuffer` ist voreingestellt. Dieses Objekt schreibt die zu speichernden Daten bis zu einer konfigurierbaren Anzahl von Bytes in den Arbeitsspeicher. Die statische Variable `maxBufferSize` wird beim Laden der Klasse auf 1 % des zu diesem Zeitpunkt verfügbaren freien Arbeitsspeichers initialisiert. Werden mehr Bytes gespeichert, wird der gesamte Datensatz in eine temporäre Datei geschrieben.

4.5.3 Methoden von OSCIDataSource

package: `de.osci.osci12.common`

Die Klasse `OSCIDataSource` definiert sechs Methoden:

- `public abstract OSCIDataSource newInstance():`

Gibt eine neue Instanz des `OSCIDataSource`-Objekts zurück.

- `public abstract java.io.OutputStream getOutpuStream():`

Gibt ein OutputStream-Objekt zurück, in welches die OSCI-Bibliothek die zu speichernden Daten schreiben kann.

- `public abstract java.io.InputStream getInputStream():`

Gibt ein `java.io.InputStream`-Objekt zurück, aus dem die gespeicherten Daten gelesen werden können. Der erste Aufruf dieser Methode muss den Schreibvorgang beenden, indem der mit `getOutputStream()` gelieferte Stream geschlossen wird. Außerdem muss der zurückgegebene `InputStream` die `reset()`–Methode so implementieren, dass nach deren Aufruf der Lesevorgang beim ersten Byte des gespeicherten Datensatzes fortgesetzt wird.

- `public abstract long getLength():`

Diese Methode liefert die Anzahl der gespeicherten Bytes.

- `public String getVersion()`

Liefert die Versionsnummer.

- `public String getVendor()`

Gibt den Namen des Herstellers zurück.

Eine detailliertere Beschreibung dieser Methoden finden sich in der JavaDoc-Dokumentation.

4.5.4 Ablauf

Muss die Bibliothek beim Verarbeiten, Versand oder Empfang einer Nachricht Daten abspeichern, so ruft sie zunächst die Methode `newInstance()` des für diesen Zweck registrierten `OSCIDataSource`-Objekts auf, um eine Instanz dieses Objekts zu erhalten. Anschließend werden die Daten in den von `getOuputStream()` gelieferten Stream geschrieben. Falls die OSCI-Bibliothek die abgespeicherten Daten im weiteren Verlauf wieder benötigt, werden sie aus dem von `getInputStream()` gelieferten Stream gelesen.

5 Programmierschnittstelle (API) der OSCI-Bibliothek

Im folgenden Abschnitt werden die Klassen, Interfaces und Resource-Dateien behandelt, die Anwendungsprogrammierer für den Betrieb der OSCI-Bibliothek benötigen.

5.1 Akteure der OSCI-Bibliothek

Die Akteure gemäß dem Rollenmodell der OSCI-Spezifikation werden in folgenden Objekten gehalten:

- `Intermed` (beschreibt einen Intermediär)
- `Originator` (beschreibt einen Sender)
- `Addressee` (beschreibt einen Empfänger)
- `Author` (beschreibt einen Autor)
- `Reader` (beschreibt einen Leser)

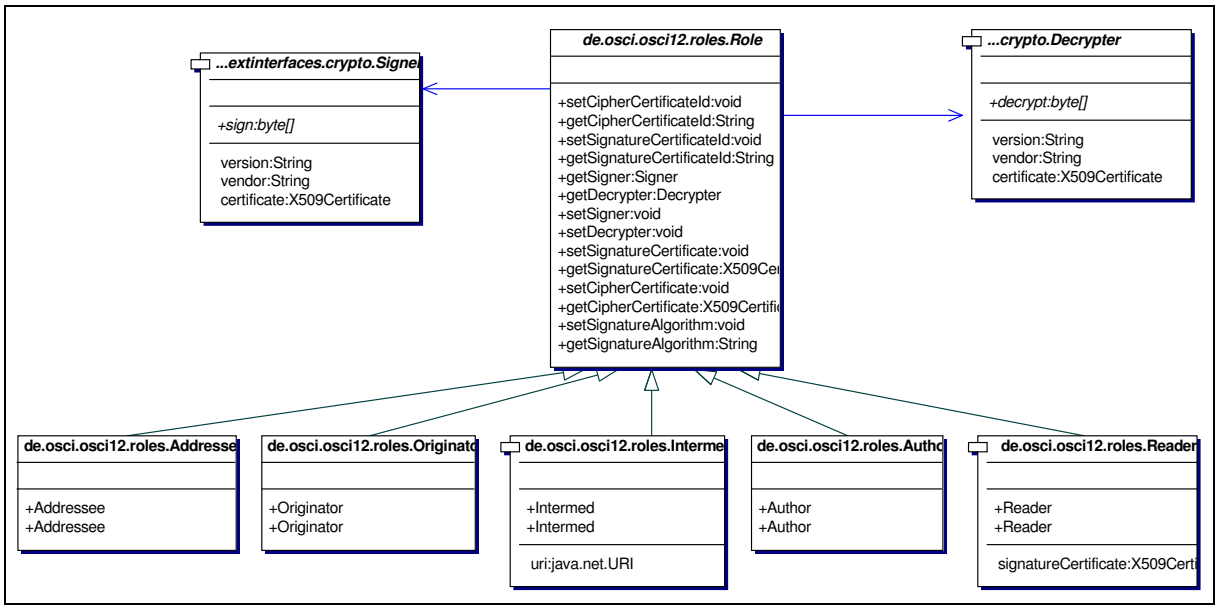

Abbildung 4: Überblick der Akteure der OSCI-Bibliothek

Die Rollen in einer OSCI-Sitzung ändern sich in den meisten Fällen nicht, sodass die Konstruktoren aller Rollen nahezu gleich aussehen. Eine Ausnahme bildet das `Reader`-Objekt, weil die OSCI-Spezifikation das Element `SignatureCertificateOtherReader` nicht verwendet und damit Signaturen des Reader (Rückantworten) nicht benötigt werden.

Es gibt zwei Möglichkeiten zum Aufbau der Rollen-Objekte:

1. Als Besitzer der privaten Schlüssel:

   Hier wird dem Konstruktor zum Signieren ein `Signer`-Objekt bzw. zum Entschlüsseln von Rückantworten ein `Decrypter`-Objekt übergeben. Beide Parameter können unter Umständen `null` sein. Bei einem `null`-Parameter wird keine Signatur bzw. Entschlüsselung gewünscht. Beispiel: `new Originator(Signer signer, Decrypter decrypter)`

2. Als Besitzer der öffentlichen Schlüssel:

   Hier wird dem Konstruktor ein X509-Signaturzertifikat zur Signaturprüfung bzw. ein X509-Chiffrierzertifikat zur Verschlüsselung übergeben. Beide Parameter können unter Umständen

`null` sein. Bei einem `null`-Parameter wird keine Signaturprüfung bzw. Verschlüsselung gewünscht.
Beispiel: `Addressee(X509Certificate signatureCertificate, X509Certificate cipherCertificate)`

5.2 Verwaltung von OSCI-Dialogen

Der Begriff des Dialogs betrifft die Auftragsebene von OSCI-Transport. Auftrag-Antwort-Paare werden zu Dialogen zusammengefasst. Jeder Dialog umfasst mindestens ein Auftrag-Antwort-Paar. Ein Dialog besteht zwischen genau einem Client und einem Supplier. Dialoge sind in der OSCI-Spezifikation von elementarer Bedeutung. Zum Beispiel muss vor dem Abholen von Nachrichten ein Dialog eröffnet werden, um die erforderliche Berechtigung zu erlangen. Die Verwaltung der Dialoginformationen über mehrere OSCI-Nachrichten hinweg übernimmt der Dialog-Handler der OSCI-Bibliothek.

5.2.1 Aufgaben des Dialog-Handler

Die OSCI-Bibliothek verwaltet für den Nutzer auch die Dialoge, das heißt, es wird die Richtigkeit von ConversationID und Sequenz-Nummer sowie die Richtigkeit der Challenge- und Responsewerte.

Folgende Aufgaben übernimmt der Dialog-Handler zur Dialogverwaltung:

- Verwaltung der OSCI-Sicherheitsmechanismen zur Absicherung des Dialogs. Dazu dienen Methoden zur Einstellung und zum Check von
  - Challenge-Response
  - Sequence-Number und Conversation-ID der Nachricht
- Festlegung konkreter Einstellungen für das jeweilige Request/Response-Tupel (bzw. auch die gesamte Session) wie
  - Attribut `checkSignatures-Signaturen` der empfangenen Nachrichten prüfen (optional für einen Client)

Zusätzlich zu den Dialog-relevanten Informationen wird der Dialog-Handler auch als Konfigurationsinstrument benutzt. Jeder OSCI-Nachricht muss im Konstruktor ein Dialog-Handler übergeben werden, sodass statische Informationen, die sich während einer Sitzung nicht ändern, ebenfalls im Dialog-Handler mitgeführt werden.

Folgende Konfigurationsinformationen werden im Dialog-Handler verwaltet:

- Konfiguration der Schnittstellen-Implementierung:
  - Transport
  - Progress Event Handler
  - DataSource

Somit werden die meisten Schnittstellen im Dialog-Handler definiert.

- Weiter werden hier die Referenzen auf folgende Akteur-Objekte einer OSCI-Kommunikation gehalten:
  - Originator (Sender der Nachricht, Client)
  - Intermediär (Supplier)

Diese Informationen werden zusammen mit der Transportschicht im Konstruktor des Dialog-Handler übergeben.

- Informationen über Transportverschlüsselung, Signatur und Spracheinstellung (`DesiredLanguages-Element`)

Jede OSCI-Request/Response-Folge erhält die Referenz auf die Dialog-Handler-Instanz, um im konkreten OSCI-Request die entsprechenden aktuellen Attributwerte setzen und die korrespondierenden Werte des jeweiligen Response auf Korrektheit gemäß OSCI-Spezifikation prüfen zu können. Nach erfolgreicher Prüfung werden die Attribute zur Dialogabsicherung jeweils sofort auf die Werte gesetzt, die für die nächste Request-Response-Sequenz gefordert sind.

Für die Initiierung einer neuen Dialogabfolge können die Attributwerte mit `resetControlBlock()` neu initialisiert werden.

In dem folgendem Schaubild werden die vom Dialog-Handler verwalteten Objekte dargestellt.

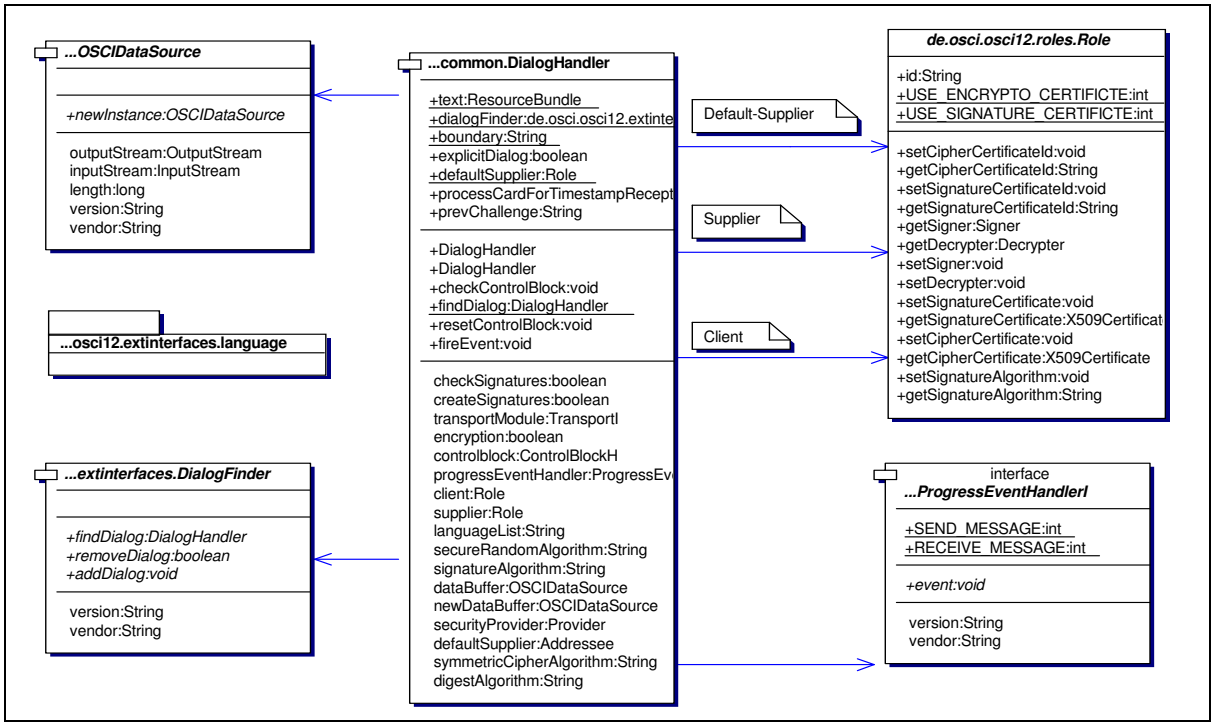

Abbildung 5: Überblick über den Dialog-Handler

5.3 OSCI -Nachrichtenobjekte

Die OSCI-Kommunikation verläuft generell in Form von Request/Response-Szenarien. Die OSCI-1.2-Spezifikation definiert insgesamt 20 verschiedene Nachrichtentypen, nämlich 10 Auftrags- und die dazugehörigen 10 Antwortnachrichten.

Für jeden Nachrichtentyp stellt die OSCI-Bibliothek eine repräsentierende Klasse bereit. Diese Klassen befinden sich in dem Package `de.osci.osci12.messagetypes`. Die gemeinsame Superklasse aller Auftragsnachrichten ist die Klasse `de.osci.osci12.messagetypes.OSCIRequest`, die der Antwortnachrichten `de.osci.osci12.messagetypes.OSCIResponseTo`. Beide Klassen sind ihrerseits von der Klasse `de.osci.osci12.messagetypes.OSCIMessage` abgeleitet.

5.3.1 Erstellen von OSCI-Nachrichten

Entsprechend den in der OSCI-Spezifikation zugewiesenen Aufgaben müssen Anwendungen OSCI-Nachrichten erzeugen können. Sie werden vom Benutzer erstellt und vom Intermediär geparst:

Nachrichtentyp	Quellcode-Dateiname
Dialoginitialisierungsauftrag	InitDialog.java
Dialogendeauftrag	ExitDialog.java
MessageID-Anforderungsauftrag	GetMessageId.java
Zustellungsauftrag	StoreDelivery.java
Zustellungsabholauftrag	FetchDelivery.java
Laufzettelabholauftrag	FetchProcessCard.java
Weiterleitungsauftrag	ForwardDelivery.java
Abwicklungsauftrag	MediateDelivery.java
Bearbeitungsantwort	ResponseToProcessDelivery.java
Annahmeantwort	ResponseToAcceptDelivery.java

Tabelle 1: Nachrichtentypen (Erstellen) und ihre zugehörigen JAVA-Klassen

Diese Klassen besitzen öffentliche Konstruktoren und können über `setX()`-Methoden mit den erforderlichen Informationen bestückt werden. Weiterhin bieten die folgenden Klassen die Möglichkeit, Inhaltsdaten in die Nachricht aufzunehmen:

- Zustellungsauftrag
- Weiterleitungsauftrag
- Bearbeitungsantwort
- Abwicklungsauftrag

Zur Aufnahme von Inhaltsdaten bedienen sich die Klassen folgender Methoden:

- `addEncryptedData(de.osci.osci12.messageparts.EncryptedData encData)`
- `addContentContainer(ContentContainer container)`

Mit Ausnahme der beiden Antwortnachrichten `ResponseToAcceptDelivery` und `ResponseToProcessDelivery`, besitzen alle Klassen außerdem eine öffentliche `send()`-Methode, mit der der Versand der Nachricht veranlasst wird. Die Methode liefert als Rückgabeparameter ein korrespondierendes Antwortnachricht-Objekt zurück.

5.3.2 Parsen von Nachrichten

Die verbleibenden Nachrichtentypen werden von Anwendungen nur als Ergebnis der `send()`-Methoden zurückgegeben (diese Nachrichtentypen werden nur vom Intermediär erstellt) oder - im Falle der beiden Auftragsnachrichten `AcceptDelivery` und `ProcessDelivery` - von der Methode `parseStream(InputStream in)` der `PassiveRecipientParser`-Klasse (passiver Empfänger)

erstellt. Die Klassen haben daher keine öffentlichen Konstruktoren. Zur Auswertung der Nachrichten stehen in den Klassen entsprechende `getX(...)`-Methoden, wie z.B. `getFeedback()`, bereit.

Folgende OSCI-Nachrichten werden vom Intermediär erstellt und vom Client geparst:

Nachrichtentyp	**Quellcode-Dateiname**
Dialoginitialisierungsantwort	ResponseToInitDialog.java
Dialogendeantwort	ResponseToExitDialog.java
MessageID-Anforderungsantwort	ResponseToGetMessageId.java
Zustellungsantwort	ResponseToStoreDelivery.java
Zustellungsabholantwort	ResponseToFetchDelivery.java
Laufzettelabholantwort	ResponseToFetchProcessCard.java
Weiterleitungsantwort	ResponseToForwardDelivery.java
Abwicklungsantwort	ResponseToMediateDelivery.java
Bearbeitungsauftrag	ProcessDelivery.java
Annahmeauftrag	AcceptDelivery.java

Tabelle 2: Nachrichtentypen (Parsen) und ihre zugehörigen JAVA-Klassen

Eingehende Beschreibungen der Methoden der speziellen OSCI-Nachrichten-Klassen finden sich im Anhang.

Bei folgenden OSCI-Nachrichtentypen können nach dem Empfang die Inhaltsdaten ausgewertet werden:

- Zustellungsabholantwort
- Annahmeauftrag
- Abwicklungsantwort
- Bearbeitungsauftrag

Zum Auswerten dieser Inhaltsdaten können die `ContentContainer` entweder mit der Methode `getContentContainer()` direkt aus der Nachricht gelesen oder – im Fall von verschlüsselten Inhaltsdaten - zuvor entschlüsselt werden. Zum Entschlüsseln eines `EncryptedData`-Objekts sind folgende Schritte notwendig:

- `EncryptedDataOSCI`-Objekt mit der Methode `getEncryptedData` aus der Nachricht extrahieren
- Analyse des verschlüsselten Datencontainers durch die Methode `getReaders();` durch diese Methode werden die Leser dieses Datencontainers bekannt gegeben.

- Auf Grundlage des `EncryptedData`-Objekts wird die Methode `decrypt(Role addressee)` aufgerufen. Als Parameter muss ein `Role`-Objekt übergeben werden, das einen `Decrypter` enthält.
- Als Ergebnis gibt die Methode `decrypt(Role reader)` ein `ContentContainer`-Objekt zurück.
- Die Auswertung des `ContentContainer` geschieht mit Hilfe der Methoden `getContents()`, `getEncryptedData()` und `getAttachments()`.

Nähere Informationen zur Weiterverarbeitung der `ContentContainer` finden sich in dem Kapitel Inhaltsdaten-Container. In dem folgenden Schaubild wird beispielhaft die Struktur der OSCI-Nachrichtenobjekte in der OSCI-Bibliothek dargestellt.

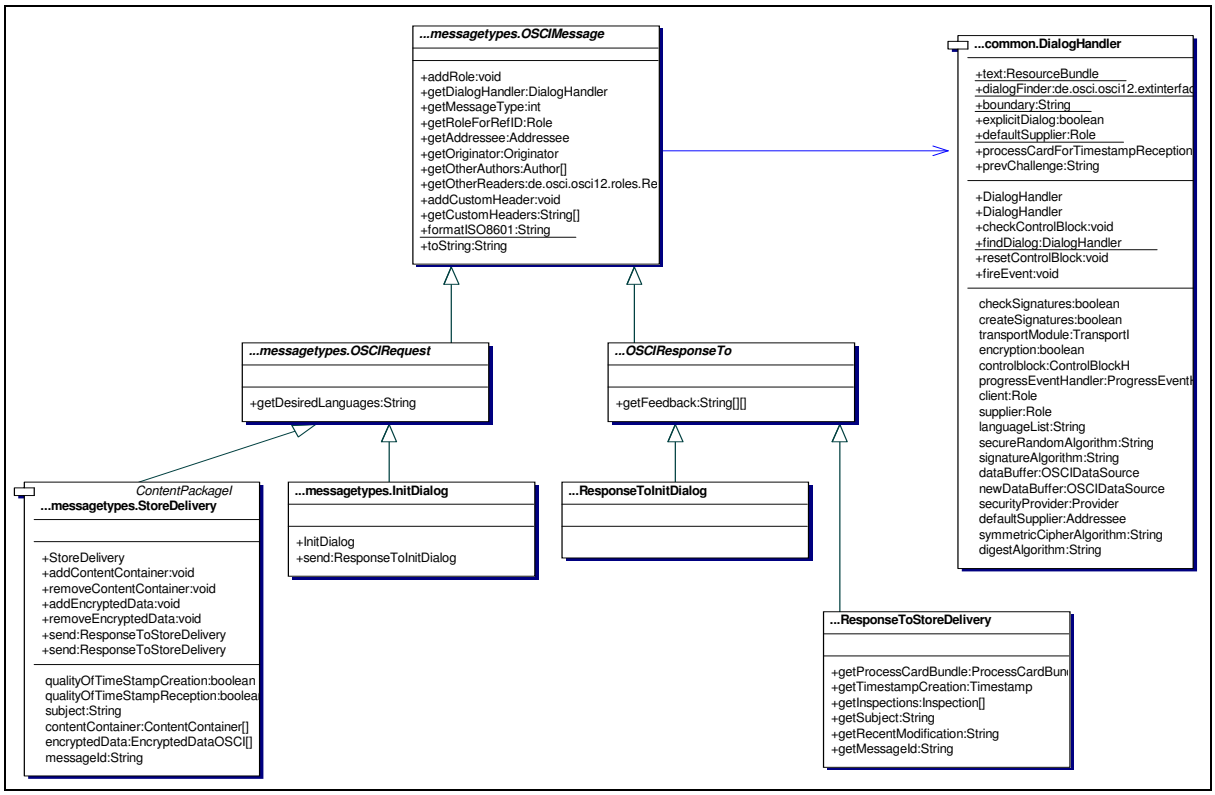

Abbildung 6: Klassendiagramm der OSCI-Nachrichtentypen

5.4 Inhaltsdaten-Container

OSCI-Nachrichten, die zur Übermittlung von Inhaltsdaten (in Form von XML-Data oder unstrukturiert als Binär-Daten) benutzt werden, enthalten immer ContentContainer zur Aufnahme dieser Informationen. Ein ContentContainer beinhaltet Content-Elemente (und ggf. EncryptedData-Elemente), die wiederum die auftragsspezifischen Inhaltsdaten aufnehmen. Auf der Empfängerseite werden die Inhaltsdaten eines ContentContainers ausgewertet. OSCI stellt ein sicheres Übermittlungsprotokoll dar, mit dem sich Inhaltsdaten in beliebiger Form (insbesondere XML-Inhaltsdaten) nachprüfbar übertragen lassen. Das Ergebnis dieser Übertragung besteht dann in der Auswertung eben dieser Inhaltsdaten im ContentContainer sowie der angebrachten Signaturen.

ContentContainer sind Einheiten, die nur als Ganzes signiert und/oder verschlüsselt werden können. Einer OSCI-Nachricht können beliebig viele dieser ContentContainer übergeben werden. Somit kann es leicht vorkommen, dass Teile (nämlich einzelne ContentContainer) einer Nachricht unverschlüsselte Informationen enthalten, während gleichzeitig andere ContentContainer derselben Nachricht schutzwürdige Inhalte aufnehmen und deswegen signiert und/oder verschlüsselt sind.

Zum leichteren Verständnis sollen einige weitere Bibliotheksklassen vorgestellt werden:

5.4.1 Content-Klasse

Content-Objekte enthalten entweder XML-Inhaltsdaten oder eine Referenz auf ein Attachment. Alle wichtigen Informationen werden schon mit dem Konstruktor gesetzt. Das Content-Objekt stellt keine Methoden zum Signieren (diese befinden sich auf ContentContainer-Ebene) oder zum Verschlüsseln (siehe `EncryptedData`-Klasse) bereit. Inhaltsdaten, seien es nun XML-Daten oder Binär-Daten, werden immer erst einmal einem Content hinzugefügt. Ggf. mehrere Content-Objekte bilden Container zum Bündeln von Inhaltsdaten. Erst auf der ContentContainer-Ebene kann die Signatur bzw. Verschlüsselung erfolgen. Die `Content`-Klasse stellt folgende Konstruktoren zur Verfügung:

- `public Content(Attachment attachment)`
- `public Content(java.io.InputStream ins)`
- `public Content(String data)`
- `public Content(ContentContainer contentContainer)`

Daneben gibt es mehrere Methoden zum Auswerten der eingestellten Daten.

Ein paar Regeln für Content-Objekte:

- Content-Objekte können nur ContentContainer zugeordnet werden.
- Content-Objekte können XML-Daten und Attachment-Referenzen enthalten.
- Für das Verschachteln von Inhaltsdaten können einem Content ContentContainer übergeben werden.

5.4.2 ContentContainer

Die `ContentContainer`-Klasse stellt einen OSCI-Auftragscontainer dar. Ein ContentContainer kann einen oder mehrere Content-(XML-Daten-)Container oder EncryptedData-Objekte enthalten. Attachments werden als Content-Objekte eingestellt, die eine Referenz auf das Attachment enthalten. Daten, die in gleicher Weise zu behandeln sind (Signieren und/oder Verschlüsseln), werden zu einem ContentContainer zusammengeführt. Sobald Inhaltsdaten unterschiedlich signiert und/oder verschlüsselt werden sollen, müssen diese auf verschiedene ContentContainer verteilt werden; die Übertragung kann zusammen in einer OSCI-Nachricht erfolgen.

5.4.2.1 Methoden zum Bearbeiten von Content-Objekten

- `public Content[] getContents()`

  Liefert die enthaltenen Content-Objekte zurück.

- `public void addContent(Content content)`

  Fügt ein Content-Objekt hinzu.

- `public void removeContent(Content content)`

  Entfernt ein Content-Objekt.

5.4.2.2 Methoden zum Bearbeiten von EncryptedData-Objekten

- `public EncryptedDataOSCI[] getEncryptedData()`

  Liefert die enthaltenen EncryptedData-Objekte.

- `addEncryptedData(EncryptedDataOSCI encryptedDataElement)`

  Fügt ein EncryptedData-Objekt hinzu.

- `public void removeEncryptedData(EncryptedDataOSCI encryptedDataElement, boolean removeAttachment)`

  Entfernt ein EncryptedData-Objekt.

5.4.2.3 Signieren eines ContentContainers

Für das Signieren eines kompletten ContentContainers steht folgende Methode zu Verfügung:

- `public void sign(Role role)`

  Diese Methode übernimmt das Signieren des kompletten ContentContainers einschließlich der in Content-Elementen referenzierten Attachments. Für Mehrfachsignaturen kann diese Methode mehrfach mit verschiedenen Rollen-Objekten aufgerufen werden.

5.4.2.4 Methoden zum Auswerten des ContentContainers

- `public Role[] getRoles()`

  Liefert alle beteiligten Rollen-Objekte (Verschlüsselung und Signatur).

- `public Role[] getSigners()`

  Liefert alle beteiligten Signatur-Rollen-Objekte.

5.4.2.5 Anbringen von seriellen-Signaturen

Serielle Signaturen können durch das Verschachteln mehrerer ContentContainer angebracht werden. Hierzu wird ein signierter ContentContainer einem Content-Objekt übergeben. Dieser Content wird dann einem weiteren ContentContainer zugeordnet, der wiederum signiert werden kann.

Aufbau dieser Konstruktion

•Content-Container (mind. einmal signiert)

- •Content(xml)
- •Content(Attachment)
- •Content
  - •Content-Container (mind. einmal signiert)
    - •Content(xml)
    - •Content(Attachment)

5.4.2.6 Regeln für ContentContainer-Objekte

- ContentContainer sind logische Einheiten zum Bündeln von Informationen.
- ContentContainer können OSCI-Nachrichten, Content-Objekten (zum Verschachteln) und EncryptedDataOSCI Objekten (für die Verschlüsselung) zugeordnet werden.
- ContentContainer nehmen Content-Objekte und EncryptedDataOSCI-Objekte (für parallele Verschlüsselung) auf.

5.4.2.7 Beispiele für das Zusammenstellen von ContentContainern

In folgendem Beispiel werden dem ContentContainer zwei Content-Objekte mit XML-Inhaltsdaten hinzugefügt.

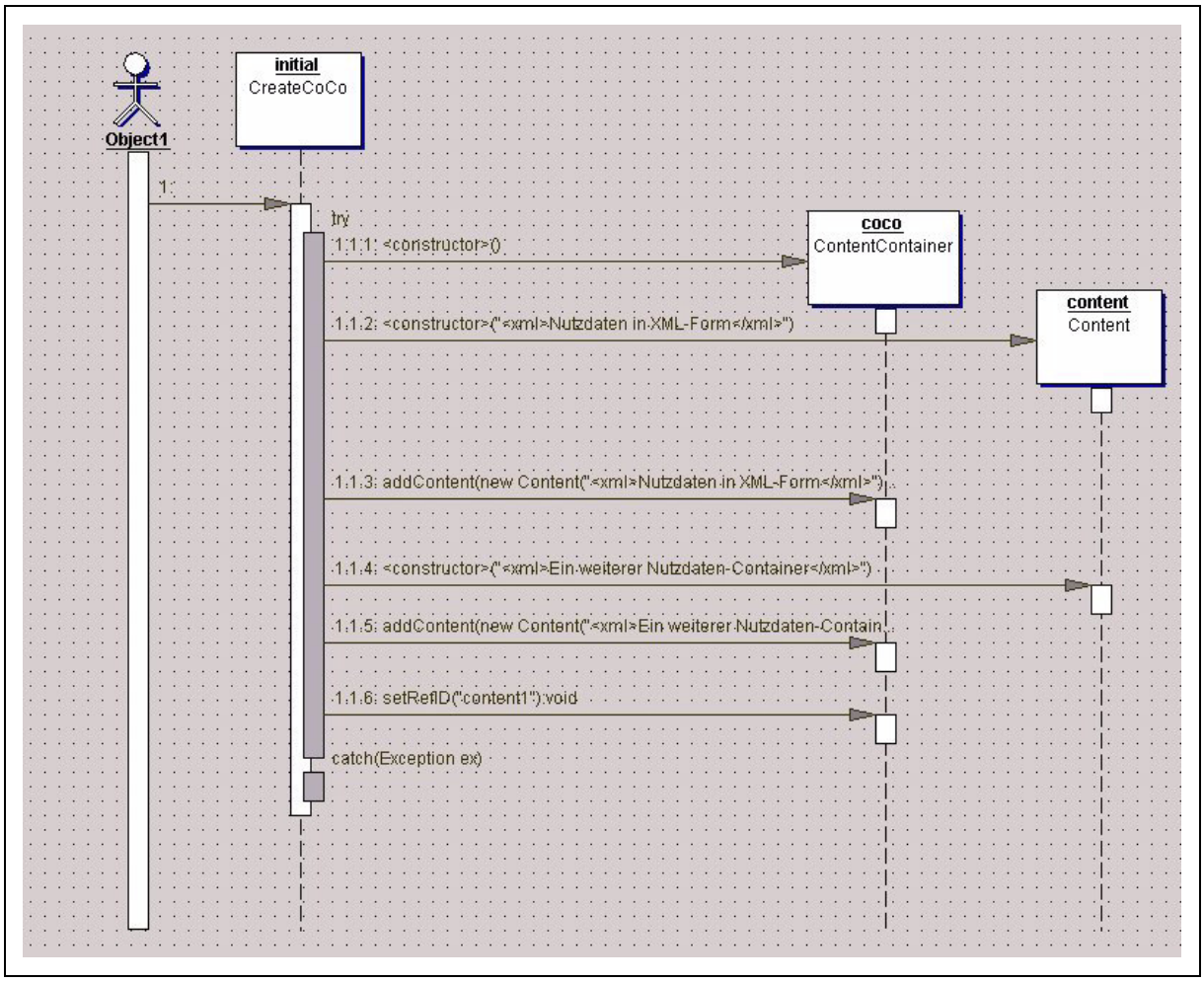

Abbildung 7: ContentContainer mit zwei Content-Objekten

In folgendem Beispiel werden ein Content-Objekt mit XML-Inhaltsdaten und ein Content-Objekt mit einem Attachment als Inhalt dem ContentContainer hinzugefügt.

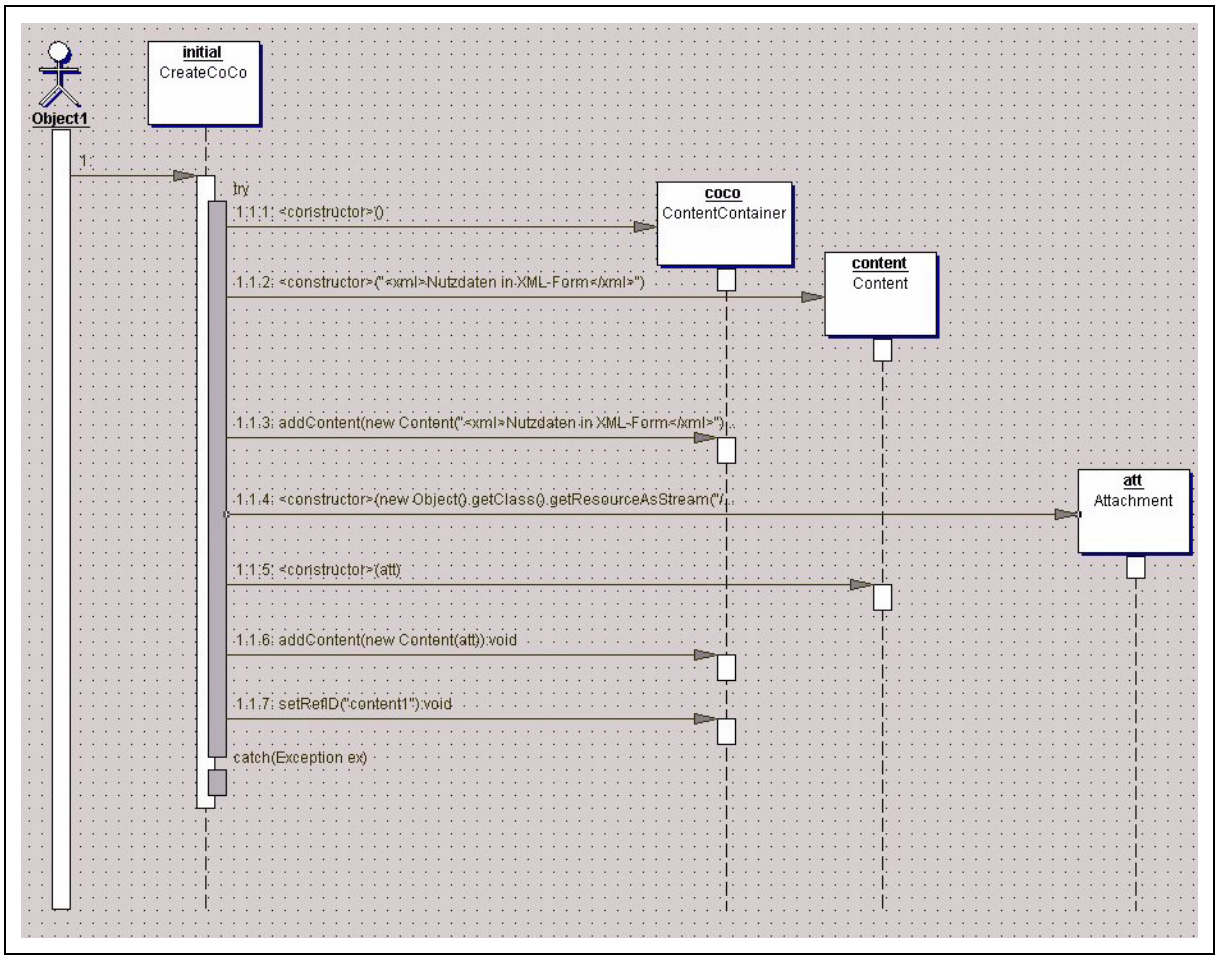

Abbildung 8: ContentContainer mit Content-Objekten und Attachment

5.4.3 EncryptedData-Klasse

Die `EncryptedData`-Klasse wird benutzt, um XML-Encryption-Elemente zu erzeugen, d.h. Inhaltsdaten zu verschlüsseln. Einer `EncryptedData`-Klasse kann ein ContentContainer übergeben werden, um diesen mit den entsprechenden Methoden zu verschlüsseln. Die `EncryptedData`-Klasse stellt einen Datencontainer für verschlüsselte Daten in einer OSCI-Nachricht dar. Auch hier werden die wichtigen Informationen beim Konstruktor übergegeben.

Die Konstruktoren sehen folgendermaßen aus:

- `public EncryptedData(javax.crypto.SecretKey secretKey, String encryptionMethodAlgorithm, ContentContainer coco)`

Dieser Konstruktor bekommt ein SecretKey-Objekt zum symmetrischen Verschlüsseln der Inhaltsdaten, den symmetrischen Verschlüsselungsalgorithmus sowie den vorbereiteten Inhaltsdatencontainer (ContentContainer) mit den eigentlichen, zu verschlüsselnden Informationen.

Durch diesen Konstruktor sind die wichtigsten Informationen gesetzt. Für das eigentliche Verschlüsseln können folgende Methoden aufgerufen werden:

- `public void encrypt(de.osci.osci12.roles.Role reader)`

Verschlüsselt den symmetrischen Schlüssel mit dem übergebenen Role-Objekt und fügt ihn als `EncryptedKey`-Element dem `EncyptedDataOSCI`-Element hinzu.

- `public void encrypt(byte[] encryptedSymKey,de.osci.osci12.roles.Role reader)`

Erzeugt einen XML-Encryption-Eintrag, ohne den symmetrischen Schlüssel noch einmal mit dem Role-Objekt zu verschlüsseln. Das Role-Objekt wird lediglich für das Einstellen der Entschlüsselungsinformationen benutzt. Es existiert bereits ein verschlüsselter symmetrischer Schlüssel.

Die `encrypt`-Methoden können mehrfach nacheinander aufgerufen werden, um für weitere Adressaten zu verschlüsseln. Die hinzugefügten Empfänger-Zertifikate werden beim Hinzufügen des EncryptedData-Objekts zur OSCI-Nachricht übertragen.

Folgendes Sequenzdiagramm verdeutlicht das Erstellen eines verschlüsselten Inhaltsdatencontainers:

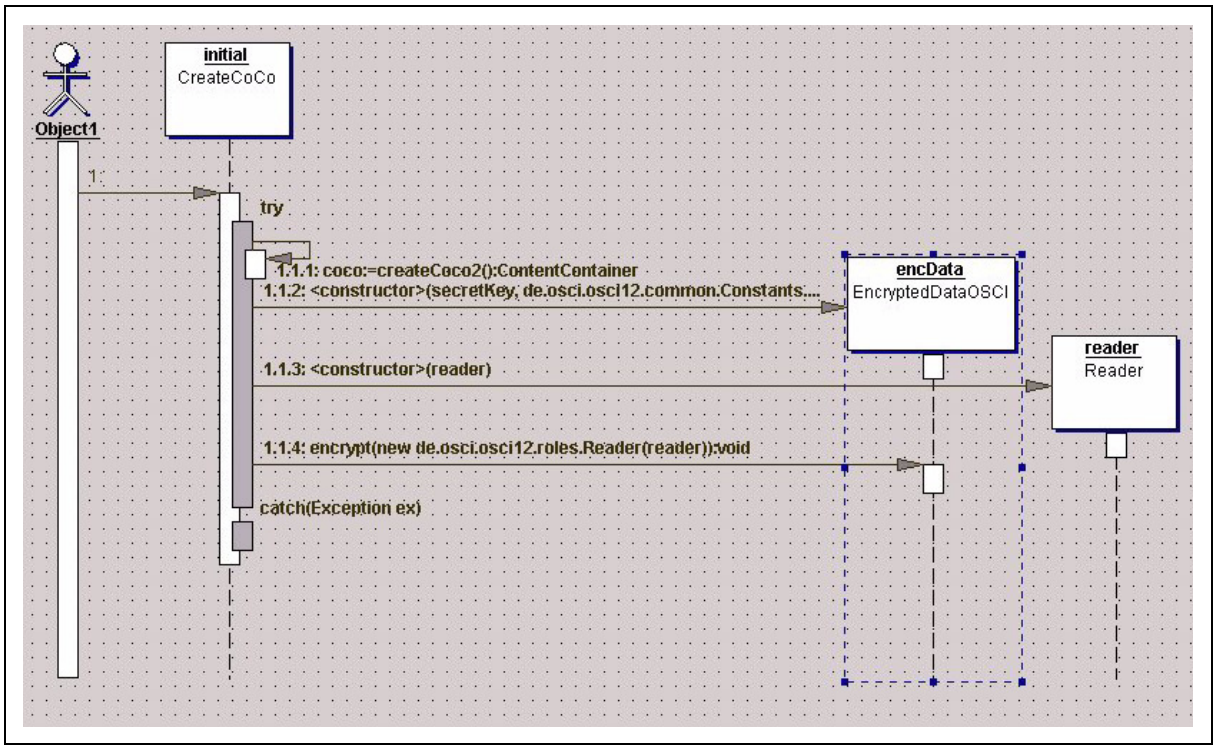

Abbildung 9: Erstellen eines verschlüsselten Inhaltsdatencontainers

Im obigen Diagramm wird dem Konstruktor von EncryptedDataOSCI ein schon bestehender ContentContainer übergeben, ein neuer Leser erstellt und die Verschlüsselung des Inhaltsdatencontainers mit dem Zertifikat des Lesers durchgeführt.

5.4.3.1 Regeln für EncryptedDataOSCI-Objekte

- EncryptedDataOSCI-Objekte können nur ContentContainer und OSCI-Nachrichten zugefügt werden.
- EncryptedDataOSCI-Objekte können nur ContentContainer aufnehmen.

5.4.4 Beispielszenarien für das Zusammenstellen von Inhaltsdaten

Exemplarisch soll der Vorgang des Hinzufügens eines signierten und verschlüsselten Inhaltsdatencontainers zu einer Nachricht vorgestellt werden:

1. Erstellen eines Content-Objekts und Übergabe der XML-Inhaltsdaten an das Content-Objekt (n-mal)
2. Hinzufügen der Content-Objekte zu dem ContentContainer

3. Signieren des ContentContainer mit dem übergebenen Author-Objekt
4. Einhängen des ContentContainer in ein EncryptedData-Objekt
5. Verschlüsseln des EncryptedData-Objekts (mit Hilfe eines Addressee-Objekts)
6. Hinzufügen des EncryptedData-Objekts zu der OSCI-Nachricht

Im Sourcecode sieht der Ablauf folgendermaßen aus:

```
// Vorbereitungen; Erstellen der Rollen-Objekte
X509Certificate empfaenger = Tools.createCertificate(new
     FileInputStream("Empfaenger.cer"));
Reader leser= new Reader(empfaenger);
Addressee addressee = new Addressee(null, empfaenger);
DialogHandler dialog=this.getDialogHandler(1);
// Erstellen der OSCI-Nachricht mit ihren Rollen-Objekten
StoreDelivery osci = new StoreDelivery(dialog, addressee);
// Erstellen eines ContentContainers
ContentContainer coco = new ContentContainer();
// Hinzufügen eines Contents
coco.addContent(new Content("<xml>Inhaltsdaten in XML-Form</xml>"));
// Signieren des Contentcontainers
coco.sign(new Author (new AuthorSigner(),null),
     );

// Erstellen eines Encrypteddata-OSCI-Objekts
EncryptedDataOSCI encData= new EncryptedDataOSCI(
  Constants.SYMMETRIC_CIPHER_ALGORITHM_TDES,coco);
// Verschluesseln der Inhaltsdaten
encData.encrypt(leser);
// Hinzufuegen der fertigen Inhaltsdaten zur OSCI-Nachricht
osci.addEncryptedData(encData);
```

Damit wurde eine einfach signierte, verschlüsselte Nachricht erstellt. Um Mehrfachverschlüsselung zu erreichen, müssen dem ContentContainer anstatt dem Content bereits verschlüsselte Nachrichten hinzugefügt werden.

Für das Versenden mit Mehrfachverschlüsselung (Vier-Augen-Prinzip) müssen folgende Schritte durchgeführt werden:

1. Erstellen eines Content-Objekts und Übergabe der XML-Inhaltsdaten an den Content (x-mal)
2. Hinzufügen der Content-Objekte zu dem ContentContainer
3. Verschlüsseln des ContentContainer mit der EncryptedData-Klasse und dem ersten Reader-Objekt
4. Hinzufügen des EncryptedData-Objekts zu einem neuen ContentContainer
5. Erstellen eines neuen Content-Objekts und Hinzufügen der Content-Objekte zu dem ContentContainer (optional)
6. Signieren des zweiten ContentContainers mit dem übergebenen Author-Objekt (optional)
7. Verschlüsseln des zweiten ContentContainers mit der EncryptedData-Klasse und dem zweiten Reader-Objekt
8. Hinzufügen des EncryptedData-Objekts zu der OSCI-Nachricht

6 Fehlerbehandlung und Fehlercodes

Entsprechend den Vorgaben der OSCI-Transport-1.2-Spezifikation wird zwischen Fehlermeldungen auf Nachrichtenebene und Rückmeldungen auf Auftragsebene unterschieden. Erstere stellen grundsätzlich echte Fehler dar. Sie werden von den auslösenden Methoden je nach Art des Fehlers durch eine der beiden folgenden Exceptions angezeigt:

- SOAPClientException
- SOAPServerException

Diese Exceptions enthalten den OSCI-Fehlercode. Die Methode `getMessage()` (bzw. `getLocalizedMessage())` liefert den dazugehörigen Text in der Sprache des default-Locale.

Bei Rückmeldungen auf Auftragsebene wird zwischen Warnungen und Fehlern unterschieden. Während Erstere (OSCI-Code beginnend mit 9) eine Exception vom Typ OSCIErrorException auslösen, wird bei Warnungen (OSCI-Code beginnend mit 3) die Verarbeitung der betreffenden Nachricht fortgesetzt. Warnungen müssen von der Anwendung selbst durch Auswertung des Feedback-Eintrags der Antwortnachricht ermittelt werden.

Fehler, die innerhalb des Bibliothekscodes auftreten, werden durch verschiedene Exceptions angezeigt. Sofern es sich nicht um Exceptions der JAVA-API handelt, sind alle diese Exceptions von OSCIException abgeleitet und erben die o.g. Implementierung der Methode `getMessage()`.

7 Aufbau der Bibliothek

Die OSCI-Bibliothek besteht aus einer Vielzahl von Klassen, die in ihrer Gesamtheit die Szenarien der OSCI-Spezifikation abdecken sollen (Client- und Supplier-seitig). Die Klassen der OSCI-Bibliothek sind alle in dem Package `de.osci.osci12` untergebracht. Unterhalb dieses Package befinden sich die untergeordneten Packages, welche die Anwendungsentwickler benutzen. In dem folgenden Schaubild wird die Package-Struktur (mit ihren Klassen) dargestellt.

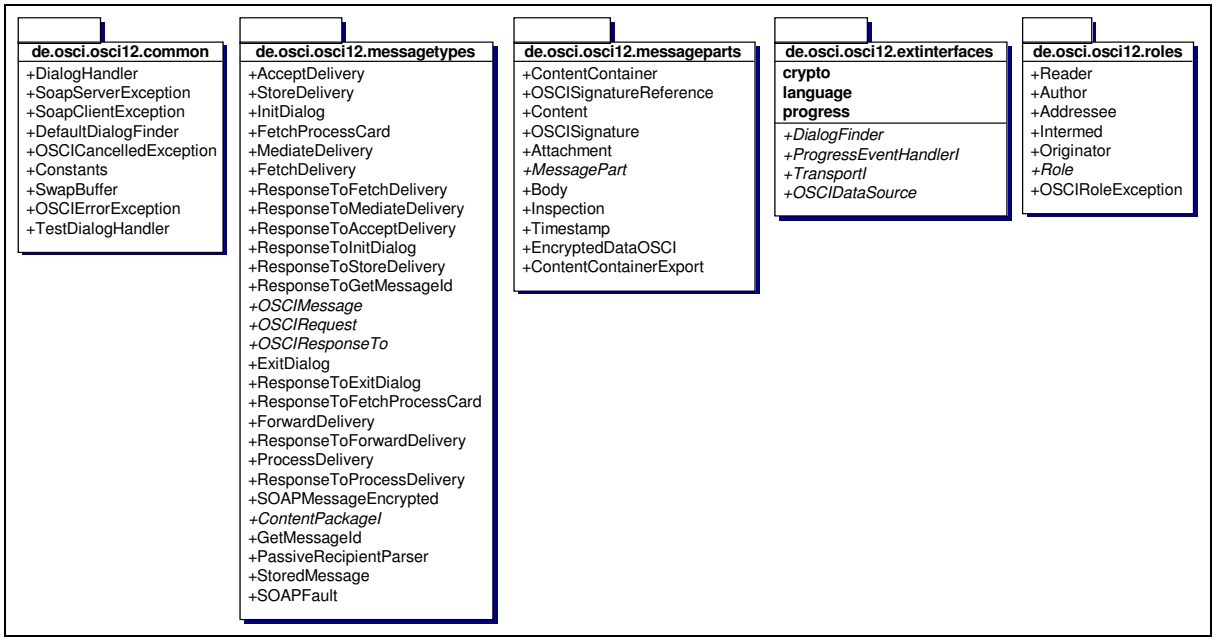

Abbildung 10: Package-Übersicht der OSCI-Bibliothek

Erläuterung zu der Grafik:

- `de.osci.osci12.common`

Dieses Package enthält allgemeine Klassen, welche zumeist von vielen anderen Klassen aus verschiedenen Packages benutzt werden. Package-übergreifende Klassen, wie z.B. der Dialog-Handler und der SwapBuffer (die Default-Implementierung der DataSource-Schnittstelle) sowie allgemeine Konstanten und Helferklassen, können in diesem Package gefunden werden.

- `de.osci.osci12.extinterfaces`

In diesem Package können alle externen Schnittstellen der Bibliothek gefunden werden. Dies betrifft die Interfaces sowie die abstrakten Klassen der OSCI-Spezifikation. Weitere Informationen zu den Schnittstellen finden sich in Kapitel 4.

- `de.osci.osci12.messageparts`

In diesem Package befinden sich Klassen, die Bestandteile von OSCI-Nachrichten sind und in vielen Nachrichten Verwendung finden. Dazu gehören etwa Repräsentationen der OSCI-Elemente Content-Container, Content, EncryptedData, Attachment, TimeStamp usw.

- `de.osci.osci12.messagetypes`

Dieses Package beinhaltet die Repräsentationen der OSCI-Nachrichtenobjekte. Für jede OSCI-Nachricht, ob Request oder Response, findet man in diesem Package eine gleichnamige Klasse - sofern sie nicht nur intermediärsseitig relevant ist. Weitere Informationen zu den OSCI-Nachrichtenobjekten finden sich in Kapitel 5.3.

- `de.osci.osci12.roles`

In diesem Package werden alle in der OSCI-Spezifikation erwähnten Akteure als JAVA-Klasse abgebildet. Weitere Informationen zu den Akteuren der OSCI-Bibliothek finden sich in Kapitel 5.1.

Weiter existieren folgende interne Packages, die vom Benutzer der OSCI-Bibliothek nicht direkt benutzt werden:

- `encryption`
- `signature`
- `soapheader`

8 Usecases

8.1 Use Case Nr. 1: Senden eines Zustellungsauftrags

Use Case Nr.	1
Use Case Name	Senden eines Zustellungsauftrags
Geschäftsvorfall	Versand einer Nachricht in Form eines OSCI-Zustellauftrags
Initiierender Aktor	Sender
Weitere Aktoren	Intermediär
Kurzbeschreibung	Versenden einer OSCI-Nachricht mit Inhaltsdatencontainer und Signaturen an den Intermediär
Vorbedingungen	Verwendete Module müssen konfiguriert sein (Crypto, Transport und Sprache). Die Informationen/Zertifikate der beteiligten Personen/Institutionen müssen eingestellt werden (Sender, Empfänger und Intermediär)
Resultate	Zustellungsantwort mit Informationen zu dem Zustellungsauftrag

1. Erstellen eines Originator-Objekts
```
ori = new Originator(new MySigner(...), new MyDecrypter(...)
```
2. Erstellen eines Intermed-Objekts
```
intermed = new Intermed(null,x509IntermedCert,"http://uri/des/intermeds"
```
3. Dialog-Handler-Objekt für die Steuerung des Dialogablaufs anlegen
```
DialogHandlerClient dh = new DialogHandlerClient
(ori,intermed,transportModul);
```
4. Erstellen eines OSCI-Nachrichtenobjekts (MessageID-Anforderung), Versenden der Nachricht und Einlesen der Rückantwort
```
GetMessageID msgIDReq = new GetMessageID(dh)
ResponseToGetMessageID msgIDRsp=msgIDReq.send()
```
5. Erstellen eines OSCI-Nachrichtenobjekts (Zustellungsauftrag)
```
StoreDelivery strDeliReq = new StoreDelivery(dh);
strDeliReq.setMsgID(msgIDRsp.getMessageID)
```
6. Erstellung eines Inhaltsdatencontainers mit Nutzdaten (XML)
```
Content content = new Content("hier nun die Nutzdaten");
ContentContainer coco = new ContentContainer ();
coco.addContent(content);
```
7. Signieren der Inhaltsdaten
```
coco.sign(new Author(...))
```

8. Inhaltsdaten dem Nachrichtenobjekt zuordnen
   ```
   strDeliReq.addContentContainer(coco);
   ```
9. Verschicken der OSCI-Nachricht
   ```
   ResponseToStoreDelivery strDeliRsp = strDeli.send()
   ```
10. Überprüfen der gerade gesendeten Nachricht
    ```
    ProcessCard proCard = strDeliRsp.getProcessCard()
    ```

Alternative Pfade	Keine
Fehlersituationen, Ausnahmen	OSCI-Fehlercodes des Typs 9nnn lösen Exceptions aus; andere Fehler, Warnungen und Informationen können den OSCI-FeedBack-Objekten der Nachricht entnommen werden.

8.2 Use Case Nr. 2: Senden eines Zustellungsabholauftrags

Use Case Nr.	2
Use Case Name	Senden eines Zustellungsabholauftrags
Geschäftsvorfall	Ein Empfänger holt eine Nachricht aus seinem Postfach ab.
Initiierender Aktor	Wegzugsmeldebehörde
Weitere Aktoren	Intermediär
Kurzbeschreibung	Abholen einer OSCI-Nachricht vom Intermediär als Empfänger/Leser
Vorbedingungen	Die verwendeten Module müssen konfiguriert sein (Crypto, Transport und Sprache). Die Informationen/Zertifikate der beteiligten Personen/Institutionen müssen eingestellt werden (Sender und Intermediär). Der Privatschlüssel des Empfängers (privateKey) muss vorhanden sein.
Resultate	Zustellungsabholantwort mit dem Inhalt des Zustellungsauftrags

1. Erstellen eines Originator-Objektsori
```
new Originator(new MySigner(...), new MyDecrypter(...)
```
7. Erstellen eines Intermed-Objektsintermed
```
new Intermed(null,x509IntermedCert,"http://uri/des/intermeds"
```
8. Dialog-Handler-Objekt für die Steuerung des Dialogablaufs anlegen
```
DialogHandlerClient dh =
new DialogHandlerClient(ori,intermed,transportModul);
```
9. Dialog-Initialisierungsnachricht erzeugen und versenden, Antwortobjekt einlesen
```
ResponseToInitDialog initDialogRsp initDiaRsp = new InitDialog(dh).send();
```
10. Zustellungsabholauftrag anlegen, versenden und Antwortobjekt einlesen
```
FetchDelivery fetchDev = new FetchDelivery(dh)
fetchDev.setReceptionOfDelivery(new Date(...))
ResponseToFetchDelivery fetchDevRsp = fetchDev.send();
```
11. Inhaltsdatencontainer extrahieren, ggf. entschlüsseln/Signatur prüfen, Daten auslesen
```
ContentContainer coco[] = fetchDevRsp.getContentContainer();
EncryptedDataOSCI encData[] = fetchDevRsp.getEncryptedData();
[n-mal]: encData[n].decrypt(new Reader(...))
[n-mal m-mal]: coco[n].getContents()[m].getContentStream();
[n-mal]: coco[n].getAttachments()
```

Alternative Pfade	Keine
Fehlersituationen, Ausnahmen	Beim Prüfen der Inhaltsdatensignaturen und beim Entschlüsseln des Inhaltsdatencontainers können Fehler auftreten, welche zu Exceptions führen. Auch beim Überprüfen des Dialog-Controlblocks werden falsche

	Dialogkontexte durch Fehlermeldungen beantwortet.

8.3 Use Case Nr. 3: Senden eines Weiterleitungsauftrags

Use Case Nr.	3
Use Case Name	Senden eines Weiterleitungsauftrags
Geschäftsvorfall	Versand einer Nachricht in Form eines OSCI-Weiterleitungsauftrags
Initiierender Aktor	Sender eines Weiterleitungsauftrags
Weitere Aktoren	Intermediär
Kurzbeschreibung	Versenden einer OSCI-Nachricht mit Inhaltsdatencontainer und Signaturen an den Intermediär. Als Ergebnis wird eine Antwort vom Fachverfahren erwartet.
Vorbedingungen	Verwendete Module müssen konfiguriert sein (Crypto, Transport und Sprache). Die Informationen/Zertifikate der beteiligten Personen/Institutionen müssen eingestellt werden (Sender, Empfänger und Intermediär).
Resultate	Weiterleitungsantwort mit Informationen zur Weiterleitung.

1. Erstellen eines Originator-Objekts
```
ori = new Originator(new MySigner(...), new MyDecrypter(...)
```
2. Erstellen eines Intermed-Objekts
```
intermed = new Intermed(null,x509IntermedCert,"http://uri/des/intermeds"
```
3. Dialog-Handler-Objekt für die Steuerung des Dialogablaufs anlegen
```
DialogHandlerClient dh =
new DialogHandlerClient(ori,intermed,transportModul);
```
4. Erstellen eines OSCI-Nachrichtenobjekts (MessageID-Anforderung), Versenden der Nachricht und Einlesen der Rückantwort
```
GetMessageID msgIDReq = new GetMessageID(dh)
ResponseToGetMessageID msgIDRsp = msgIDReq.send()
```
5. Erstellen eines OSCI-Nachrichtenobjekts (Weiterleitungsauftrag)
```
ForwardDelivery strForwReq = new ForwardDelivery(dh);
strForwReq.setMsgID(msgIDRsp.getMessageID)
```
6. Erstellung eines Inhaltsdatencontainers mit Nutzdaten (XML)
```
Content content = new Content("hier nun die Nutzdaten");
ContentContainer coco=new ContentContainer ();
coco.addContent(content);
```
7. Signieren der Inhaltsdaten
```
coco.sign(new Author(...))
```
8. Inhaltsdaten dem Nachrichtenobjekt zuordnen

strDeliReq.addContentContainer(coco);

9. Verschicken der OSCI-Nachricht
   ```
   ResponseToForwardDelivery strForwRsp= strForwReq.send()
   ```
10. Überprüfen der gerade gesendeten Nachricht
    ```
    ProcessCard proCard = strForwRsp.getProcessCard()
    ```

Alternative Pfade	Keine
Fehlersituationen, Ausnahmen	OSCI-Fehlercodes des Typs 9nnn lösen Exceptions aus; andere Fehler, Warnungen und Informationen können dem OSCI-FeedBack-Objekten der Nachricht entnommen werden.

9 Abbildungsverzeichnis

10 Tabellenverzeichnis