technische universität
dortmund

Proseminararbeit

# CARiSMA

**Volkan Gümüş**
**31. Januar 2014**

Veranstalter: Dr. Thomas P. Ruhroth

Prof. Dr. Jan Jürjens Lehrstuhl 14 Software Engineering
Fakultät Informatik
Technische Universität Dortmund
Otto-Hahn-Straße 14
44227 Dortmund
http://www-jj.cs.uni-dortmund.de/secse

Volkan Gümüş
volkan.guemues@tu-dortmund.de
Matrikelnummer: 148509
Studiengang: Bachelor Informatik

Proseminar: Werkzeugunterstützung für sichere Software
Thema: CARiSMA

Eingereicht: 31. Januar 2014

Betreuer: Jens Bürger

Prof. Dr. Jan Jürjens Lehrstuhl 14 Software Engineering
Fakultät Informatik
Technische Universität Dortmund
Otto-Hahn-Straße 14
44227 Dortmund

# Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Dortmund, den 31. Januar 2014

Volkan Gümüş

# Inhaltsverzeichnis

**1 Einleitung** **1**
1.1 Motivation und Hintergrund . . . . . 1
1.2 Problemstellung . . . . . 1
1.3 Ziele . . . . . 2

**2 Grundlagen** **3**
2.1 UML und UMLsec . . . . . 3
2.1.1 UML . . . . . 3
2.1.2 Erweiterungsmechanismen von UML . . . . . 3
2.1.3 UMLsec . . . . . 3
2.2 Object Constraint Language . . . . . 4
2.3 Business Process Model and Notation . . . . . 4

**3 Das CARiSMA-Tool** **8**
3.1 Security Checks . . . . . 8
3.1.1 UML2 Modell: Static Check . . . . . 9
3.1.2 BPMN Modell: BPMN2 OCL Check . . . . . 14
3.2 Funktionalität . . . . . 16
3.3 Komponentenüberblick . . . . . 17

**4 CARiSMA: Analyse eines Beispiels** **18**

**5 Fazit** **20**
5.1 Zusammenfassung . . . . . 20
5.2 Ausblick . . . . . 20

# 1 Einleitung

## 1.1 Motivation und Hintergrund

Im Rahmen einer Softwareentwicklung muss die zu entwickelnde Software spezifiziert, konstruiert und dokumentiert werden. Solch eine abstrahierende Beschreibung lässt sich durch eine Modellierungssprache erstellen. Eine Modellierungssprache kennt Spracheinheiten, die wichtig für die meisten Modellierungen sind und stellt zwischen diesen eine Beziehung her. Eine grafische Modellierungssprache definiert zudem eine grafische Notation um Spracheinheiten oder zusammenhängende Strukturen von Spracheinheiten darstellen zu können. Eine Softwareentwicklung könnte beispielsweise durch die bekannte Unified Modeling Language folgendermaßen strukturiert werden:

- Auftraggeber prüfen und bestätigen Anforderungen an die Software in einem Anwendungsfalldiagramm.
- Arbeitsabläufe sind in einem Aktivitätsdiagramm beschrieben, welche vom Softwareentwickler anschließend umgesetzt werden.
- Installation der Software anhand eines Installationsplans in Form eines Verteilungsdiagramms.

Eine Modellierungssprache mit präzise definierter Semantik heißt Spezifikationssprache. Für die konkreten Einsatzfelder 'Geschäftsprozesse' und 'Arbeitsabläufe' ist die Business Process Model and Notation (nachfolgend BPMN) eine solche Spezifikationssprache. Diese erlaubt die Modellierung der in der Geschäftsprozessmodellierung üblichen Differenzierung in die Kontrollflussperspektiven, die Datensicht und die Sicht auf Ressourcen.

## 1.2 Problemstellung

Eine Software kann bereits während ihrer Modellierung auf sicherheitskritische Aspekte untersucht werden. Dies geschieht dadurch, dass dem zugrunde liegenden Modell Sicherheitsanforderungen zugewiesen werden. Anschließend kann überprüft werden, ob das Modell alle geforderten Sicherheitanforderungen einhält. Eine Lücke lässt sich somit frühzeitig während der Modellierungsphase erkennen und korrigieren. Die Überprüfung lässt sich durch dafür geeignete Software durchführen. Prominentes Beispiel, auf das im folgenden in der Ausarbeitung eingegangen wird, ist das

CARiSMA [3] Tool. Dieses untersucht unter anderem UML und BPMN Modelle auf die Einhaltung der Sicherheitsanforderungen, wobei weitere Modellierungssprachen durch entsprechende Plugininstallation ebenfalls unterstützt werden.

## 1.3 Ziele

Ziel der Ausarbeitung ist es, die Funktionsweise des Tools zu schildern und eine Einführung in die Benutzung zu geben, sodass das Tool verstanden und für Analysezwecke genutzt werden kann. Dafür werden die relevanten Grundlagen der Modellierungssprachen UML (Kap. 2.1.1) und UMLsec (Kap. 2.1.3), OCL (Kap. 2.2) und BPMN (Kap. 2.3) erklärt und mit Beispielen veranschaulicht. Anschließend wird eine Einführung in gängige Security Checks (Kap. 3.1), mit denen CARiSMA arbeitet, gegeben. Anschließend wird eine eine Einführung in die Benutzung des Tools gegeben, sodass die im letzten Teil (Kap. 4) gezeigten beispielhaften Ausführungen des Tools verstanden und interpretiert werden können.

# 2 Grundlagen

## 2.1 UML und UMLsec

### 2.1.1 UML

Die *Unified Modeling Language* ist eine grafische Modellierungssprache zur Spezifikation von Modellen von Softwaresystemen, welche im Jahr 1990 erschienen ist [6]. Die UML findet in vielen Entwicklungsbereichen Verwendung, eröffnet vielfältige Möglichkeiten für hoch-qualitative industrielle Entwicklungen von kritischen Systemen und berücksichtigt dabei auch die Komplexität aktueller Systeme. Die UML legt einige Bezeichnungen fest, die für diverse Modellierungen von Bedeutung sein könnten. Zwischen diesen Bezeichnern werden Relationen hergestellt, es sind zudem eine Vielzahl von graphischen Notationen und Strukturen vordefiniert.
Aktuell ist die UML in der Version 2.4.1 verfügbar und wird von der *Object Management Group* (OMG) entwickelt.
Die UML definiert zwei Gruppen von Diagrammen: Die Struktur- und die Verhaltensdiagramme, wobei jeder Gruppe 7 verschiedene Diagramme zugehören. Es sind ferner Spracheinheiten wie Klassen oder Aktionen verfügbar, mit denen die Modelle spezifiziert werden können.

### 2.1.2 Erweiterungsmechanismen von UML

Die UML lässt sich durch die Erweiterungsmechanismen *Stereotype, Tagged Values,* und *Constraints* erweitern.
Über die Stereotypes lassen sich neue Komponenten erstellen, die neue Funktionen erfüllen sollen. Diese Bauteile werden allerdings von vorhandenen Bauteilen abgeleitet und auf ihre neue Funktion spezifiziert. Tagged Values sind Eigenschaften, mit denen die Stereotypen erweitert werden können. So können zusätzliche Informationen einfach dem Stereotype hinzugefügt werden. Durch Constraints lässt sich die Semantik der UML Bausteine modifizieren, was bedeutet, dass einem Baustein neue Regeln hinzugefügt oder bestehende modifiziert werden können.
Diese Mechanismen nutzt die UML Erweiterung UMLsec um einen Ansatz für sichere Systeme zu erstellen.

### 2.1.3 UMLsec

In UMLsec werden die Erweiterungsmechanismen aus obigem Abschnitt genutzt, um dem Modell sicherheitsrelevante Anforderungen hinzuzufügen. Stereotypes wer-

den zusammen mit tags verwendet, um sicherheitsrelevante Anforderungen über die zugrundeliegende Umgebung zu formulieren. Die Kriterien, ob ein Modell die Anforderungen erfüllt, liefern die constraints. Die Stereotypes werden durch doppelte Spitzklammern << >> an das zu erweiternde Modellelement angebracht. Eine Liste der möglichen Stereotypes findet sich in [2].

## 2.2 Object Constraint Language

Die Object Constraint Language (OCL) ist eine formale Sprache, durch die Bedingungen an eine Modellierung festgelegt werden können. OCL ist so konzipiert, dass es zur Spezifikation von UML Modellen dient, jedoch nicht eigenständig zur Modellierung geeignet ist. Diese Bedingungen gehen direkt in den zu generierenden Code ein, da sie wesentliche Merkmale darstellen.

In den OCL Spezifikationen [4] werden sieben Arten von Bedingungen vorgestellt:

- *Body Definition* von seiteneffektfreien Operationen (isQuery = true).
- *Definition* von Operationen und Attributen, die nicht im Modell enthalten sind.
- *Guards* müssen gelten, wenn ein Zustandsübergang beginnt.
- *Initial and derived values* sind Bedingungen für Ausgangs- und abgeleitete Werte.
- *Invariants* müssen zu jeder Zeit für eine Instanz gelten.
- *Pre/Post-Conditions* müssen genau dann gelten, wenn die Ausführung der zugehörigen Operation beginnt bzw. endet.

## 2.3 Business Process Model and Notation

Die Business Process Model and Notation (BPMN) ist eine Modellierungssprache zur Spezifikation von Geschäftsprozessen. Im Jahr 2011 wurde BPMN 2.0 von der OMG veröffentlicht [7].
Diagramme aus BPMN werden durch *BPD* abgekürzt. Solch ein BPD soll die Entwicklung eines Prozesses (unter menschlichen Experten) unterstützen, wobei ein Prozess als eine Abfolge von Aktivitäten zu sehen ist. Diese Prozesse können durch Menschen oder Maschinen ausgeführt werden.
Ein BPD wird durch verschiedene BPMN-Basiselemente, sogenannte Symbole, aufgebaut, die in verschiedene Kategorien unterteilt und nachfolgend vorgestellt werden.

### Flow Objects:

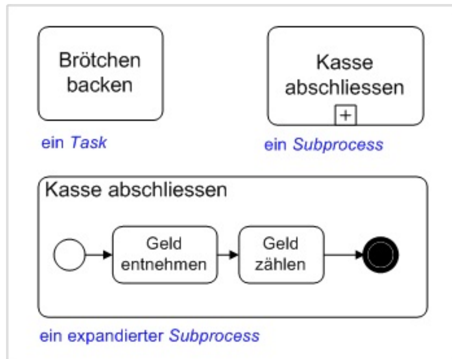


**Abbildung 2.1:** Beispiel Activities nach: [5]

Eine Aufgabe aus einem Geschäftsprozess wird durch ein *Activity*-Element der Flow Objects visualisiert, wobei es als Rechteck mit abgerundeten Ecken dargestellt wird (vgl. Abb. 2.1). Eine einfache Activity bezeichnet man als Task, komplexere Activities sind sogenannte Subprocesses. Diese können weitere Activities beinhalten. Die grafische Notation unterscheidet sich in dem Rechteck-Plus-Symbol. Ein Subprocess kann jedoch auch expandiert dargestellt werden. In diesem Fall entfällt das Rechteck-Plus-Symbol.

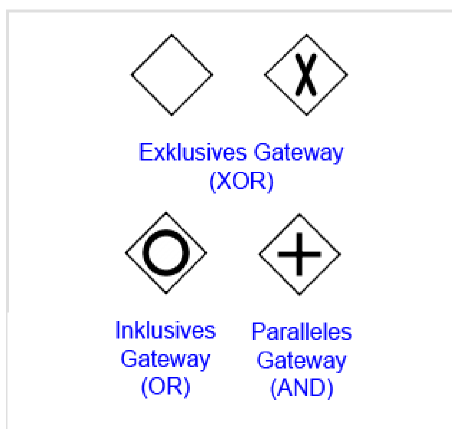


**Abbildung 2.2:** Beispiel Gateways nach: [5]

Ein Punkt, an dem verschiedene Kontrollflüsse zusammen laufen und folglich eine Entscheidung bezüglich des weiteren Verlaufs getroffen werden muss, nennt man Gateway. Solch ein Gateway wird durch ein auf der Spitze stehendes Quadrat visualisiert (vgl. Abb. 2.2) und kann ein Symbol beinhalten, was je nach Verwendung mit den logischen Werten AND, OR oder XOR zu assoziieren ist.

**Abbildung 2.3:** Beispiel Events nach: [5]

Für die Modellierung von Ereignissen, beispielsweise das Eintreffen einer Mitteilung, nutzt man Events (vgl. Abb. 2.3). Diese werden durch Kreissymbole visualisiert, wobei je nach Klasse des Events ein zusätzliches Symbol im Kreis zu finden ist. Die erste Klasse der Events bildet das Start- Intermediate- und End-Event, wobei die Position im Geschäftsprozess das wesentliche Merkmal ist. Die zweite Gruppe bilden die Events, dessen Wirkung wesentliches Merkmal ist. Dazu gehört das Catching- und das Throwing-Event. Die letzte Klasse bilden die Events, die eine direkte Zugehörigkeit haben wie beispielsweise Exception- oder Timer-Events.

**Connecting Objects:**

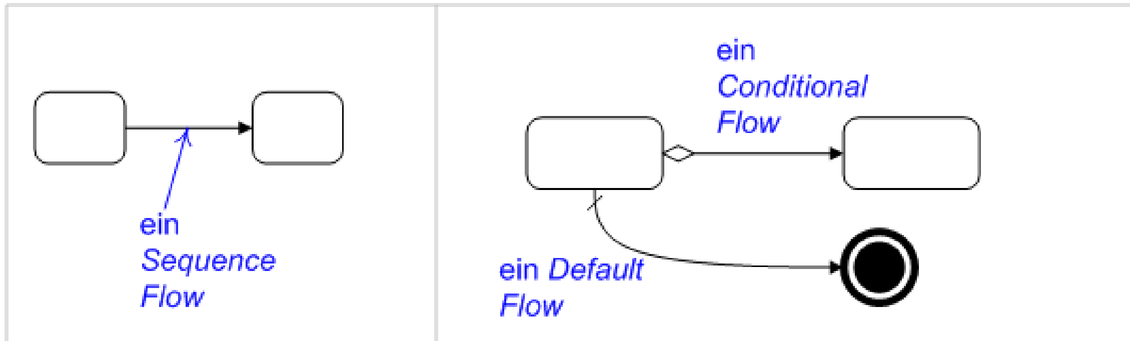


**Abbildung 2.4:** Beispiel Sequence Flows nach: [5]

Ein *Sequence Flow* ist eine Verbindung zwischen *Flow Objects*, wobei die Visualisierung durch einen Pfeil von einem zum anderen Objekt erfolgt (vgl. Abb. 2.4). Eine Verbindung, die nur durchlaufen werden soll, wenn eine bestimmte Bedingung erfüllt ist, nennt sich *Conditional Flow*. Analog dazu gibt es einen *Default Flow*, der nur durchlaufen wird, wenn kein anderer *Sequence Flow* durchlaufen werden kann. Leicht erkenntlich ist nun, dass *Sequence Flows* die Reihenfolge des Ablaufs bestimmen.

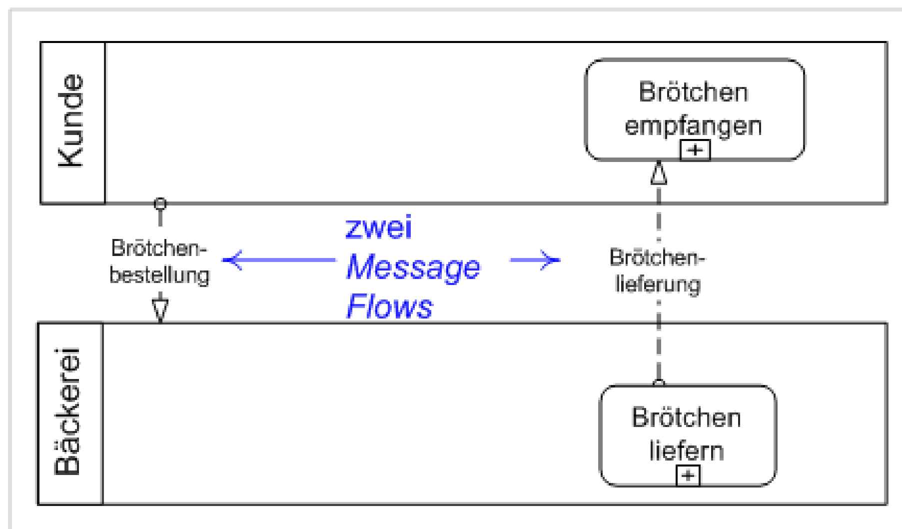


**Abbildung 2.5:** Beispiel Message Flows nach: [5]

Möchte man einen Austausch von Daten zwischen *Flow Objects, Lanes* oder *Pools* visualisieren, so nutzt man einen gestrichelten Pfeil (vgl. Abb. 2.5). Dieser wird als *Message Flow* bezeichnet.

**Swimlanes, Lanes und Pools:**

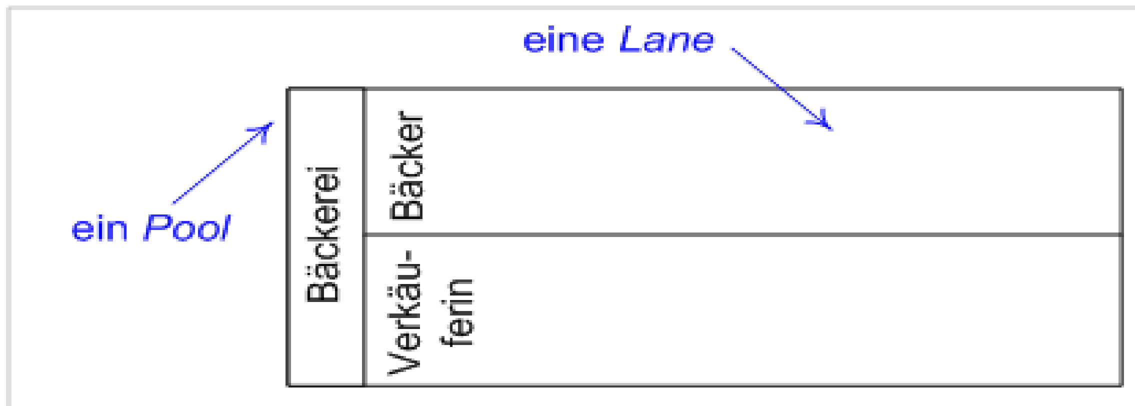


**Abbildung 2.6:** Beispiel Swimlanes nach: [5]

Für die Visualisierung eines Ablaufs innerhalb eines Prozesses nutzt man *Swimlanes* (vgl. Abb. 2.6). Swimlanes lassen sich in *Pools* und *Lanes* unterteilen, wobei durch *Pools* Beteiligte innerhalb eines Ablaufs abgegrenzt werden (vgl. Abb. 2.6 'Bäcker' und 'Verkäufer'). Eine *Lane* unterteilt einen *Pool* um eine zusammenhängende Struktur visualisieren zu können.

**Artifacts:**

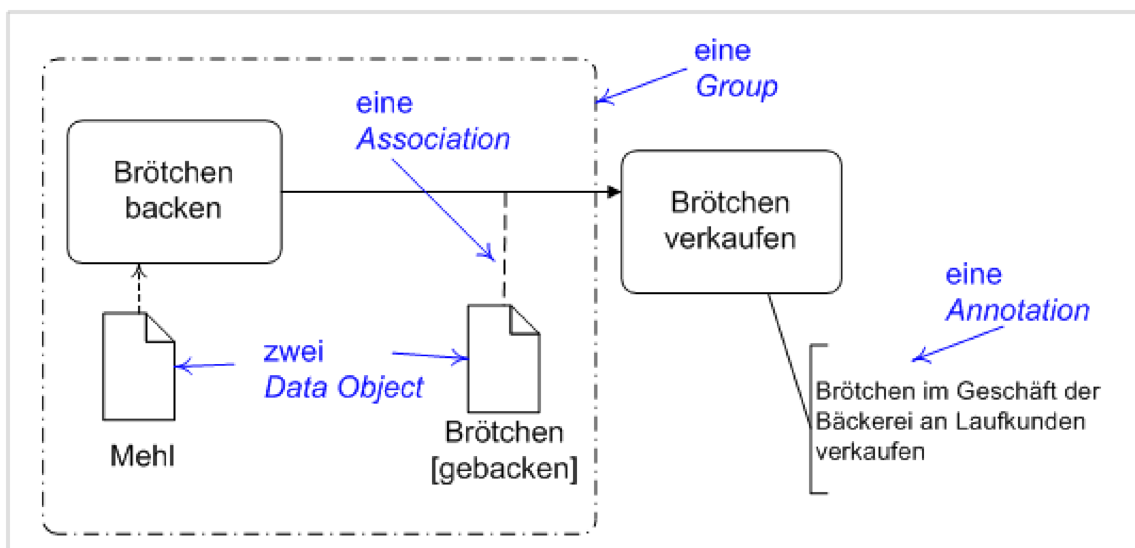


**Abbildung 2.7:** Beispiel Artifacts nach: [5]

Zusätzliche Informationen lassen sich in einem Diagramm durch *Artifacts* darstellen, wobei zwischen *Annotations, Data Objects* und *Groups* unterschieden wird (vgl. Abb. 2.7). Zur Kommentierung oder Erklärung können *Annotations* genutzt werden, wobei sich diese auf einzelne Elemente oder auch komplexe Strukturen beziehen können. Eine *Group* fasst einen Prozess **visuell** zusammen. *Data Objects* sind vom Prozess gepflegte *Artifacts*, diese können virtuelle Daten oder reale Objekte darstellen.

# 3 Das CARiSMA-Tool

Zusammenfassend lässt sich nach Kapitel 2.1.3 sagen, dass ein UMLsec Modell durch *Stereotypes, Tags* und *Constraints* Sicherheitseigenschaften enthält, die von einer zu modellierenden Systemumgebung vorausgesetzt werden.

Im Jahr 2001 wurde das Werkzeug *UMLsecTool* veröffentlicht, das genau solche UMLsec-Modelle auf die Einhaltung von Sicherheitsanforderungen untersuchen sollte. Das UMLsecTool wurde stets weiter entwickelt, wobei viele Programmierer an der Entwicklung, oftmals in Form von Arbeiten [8], beteiligt waren. Diese Entwicklung verlief jedoch unkontrolliert und schwach dokumentiert.

Das UMLsecTool wurde im Bezug auf die Funktionen weiterentwickelt, jedoch musste jede zusätzliche Funktionalität (*Plug-Ins*) fest eingebaut werden, da das Tool nicht auf eine erweiterbare Architektur aufbaut. Die Funktionalität basiert auf Bibliotheken, die alt sind und nicht mehr weiterentwickelt werden, wie beispielsweise UML 1.4.

Diese Komplikationen innerhalb des UMLsecTools resultierten zu Schwierigkeiten in der Wartung und Entwicklung, sodass eine zwingend massive Änderung des Tools notwendig wurde. Die zweite Version des UMLsecTool wurde unter dem Namen CARiSMA veröffentlicht. Die neue, Metamodell unabhängige, Grundlage bildet das *Eclipse Modeling Framework* [9]. CARiSMA unterstützt die UML Version 2.3.1, da es auf das *Eclipse Modeling Framework* aufbaut [3][9]. Beispielsweise kann CARiSMA auf ähnlich einfache Weise mit BPMN 2.0 umgehen.

Eine neue Sprache lässt sich aufgrund der Pluginverwaltung, die die neue Architektur bereitstellt, zu CARiSMA migrieren. Dafür muss das Metamodell der Sprache erstellt und ein Plugin entworfen werden, dass die Analyse auf dem Metamodell durchführt [10].

## 3.1 Security Checks

CARiSMA hat eine Vielzahl von Checks, die in die Gruppen *conformance, risk* und *security check* geteilt und in Form von Plugins bereitgestellt werden. Diese können in CARiSMA geladen und auf das zugrunde liegende Modell angewendet werden. Im folgenden werden drei populäre UML2-Checks und ein BPMN2-Check vorgestellt.

### 3.1.1 UML2 Modell: Static Check

**Fair Exchange**

Eine Transaktion zwischen Käufer und Verkäufer muss gewisse Anforderungen erfüllen, sodass ein betrügerisches Verhalten abgewendet wird. Diese Anforderungen werden durch das Stereotyp <<*fair exchange*>> beschrieben. Die Situation zwischen Käufer und Verkäufer lässt sich grob durch folgendes Anwendungsfalldiagramm beschrieben

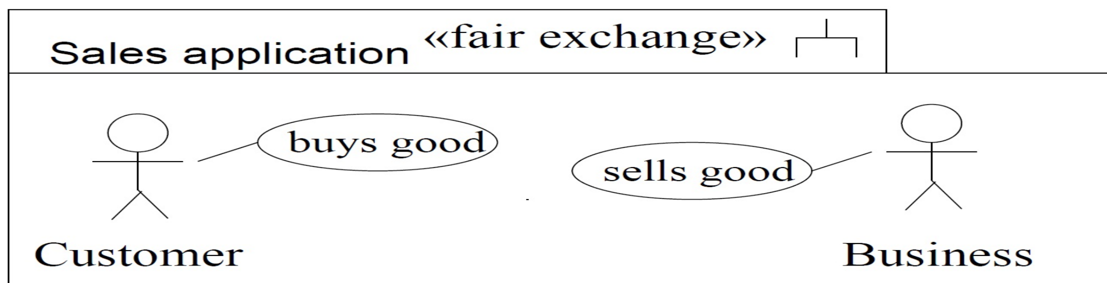


**Abbildung 3.1:** Anwendungsfalldiagramm Käufer-Verkäufer [2]

wobei die Anforderungen in den Prozessflüssen *buys good* und *sells good* definiert sind. Ein wichtiges Merkmal von 3.1 ist, dass das Diagramm mit dem Stereotyp <<*fair exchange*>> markiert ist. Die Prozesse 'buys good' und 'sells good' können durch ein weiteres Diagramm, genauer mit einem Aktivitätsdiagramm, spezifiziert werden. Das Aktivitätsdiagramm muss ebenfalls mit dem Stereotyp <<*fair exchange*>> markiert sein. Das folgende Diagramm beschreibt dies detaillierter:

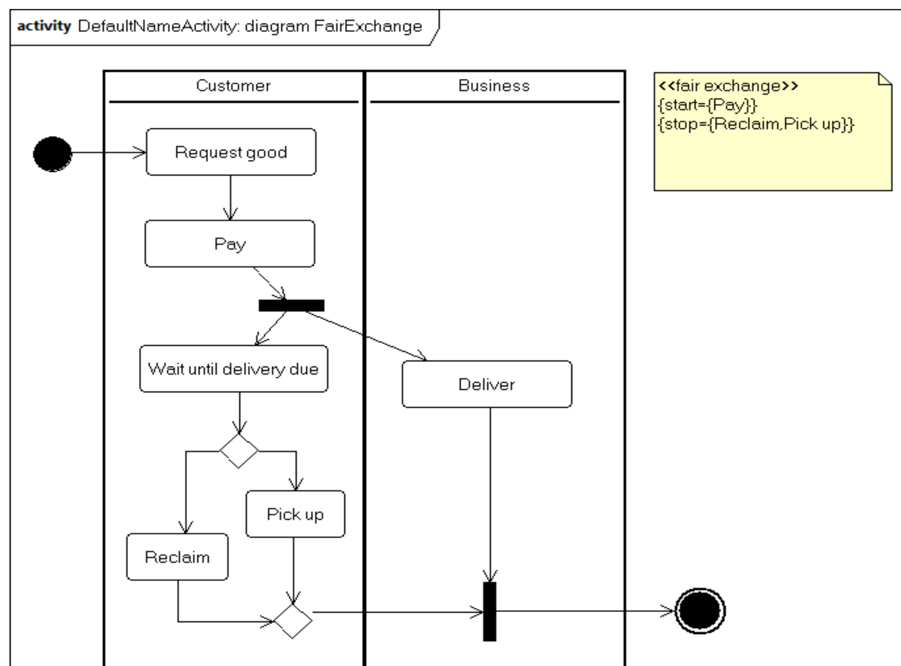


**Abbildung 3.2:** Aktivitätsdiagramm Käufer-Verkäufer [3]

Der Tag {*start*} beschreibt den Beginn, {*stop*} das Ende der Transaktion. Das obige Beispiel zeigt demnach eine beginnende Transaktion, sobald die Bezahlung für die georderte Ware beim Verkäufer eingegangen ist. Für den Verkäufer ist folglich sichergestellt, dass die Lieferung erst und nur dann erfolgt, wenn das Geld eingetroffen ist. Der Kunde hat die Möglichkeit, die Ware bei Lieferung anzunehmen oder bei nicht gelieferter Ware die Bestellung zu reklamieren. Sobald einer der beiden Fälle eingetroffen ist, kann die Transaktion beendet werden.

### Secure Dependency

An Systeme, die untereinander Daten austauschen, können sicherheitsrelevante Anforderungen vorausgesetzt werden. Der Stereotyp <<*secure dependency*>> wird zur Bezeichnung von Subsystemen, die Objektdiagramme oder Diagramme von statischen Strukturen enthalten, benutzt. Das Stereotyp <<*critical*>> erlaubt eine Verfeinerung der Sicherheitsanforderungen.
Seien $C$ und $D$ zwei Objekte, zwischen denen <<*call*>> und <<*send*>> Abhängigkeiten existieren. Diese müssen die Anforderungen auf den Daten, die durch die Tags {*secrecy*} und {*integrity*} gestellt werden, einhalten. Besteht eine <<*call*>> oder <<*send*>> Abhängigkeit von $C$ zu $D$, so ist eine Nachricht {*secrecy*} in C genau dann wenn sie {*secrecy*} in D ist.
Das Beispiel 3.3 zeigt ein System zur Erzeugung von Schlüsseln, ein sogenanntes Key Generation-Subsystem. Der Key Generator liefert einen Schlüssel, wobei der 'Random generator' diesen erzeugt. Key Generator stellt eine Sicherheitsanforderung an *newKey()* und *random()*, wobei Random generator keine sicherheitsrelevanten Informationen zu random() gibt. Damit ist <<*secure dependency*>> nicht erfüllt.

### Secure Links

Das Beispiel aus 3.2 lässt sich erweitern: Der Verkäufer betreibt einen Online-Shop, in dem er die angesprochene Ware verkauft. Will der Käufer diese Ware erwerben, so muss er über ein internetfähiges Endgerät den Online-Shop aufrufen und die

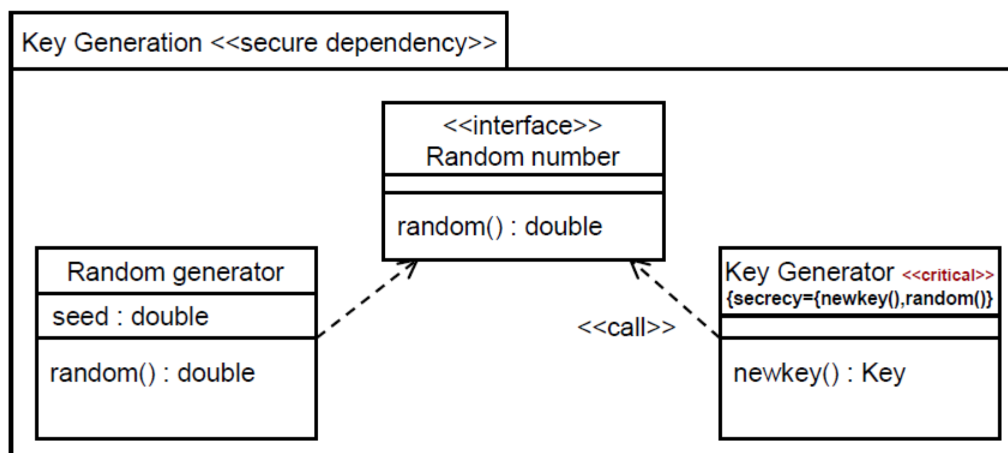


**Abbildung 3.3:** Beispiel zu <<*secure dependency*>> [2]

Bestellprozedur durchlaufen. Die Anforderungen an die physikalische Schicht, über die diese Kommunikation abläuft, stellt das Stereotyp <<*secure links*>> dar.

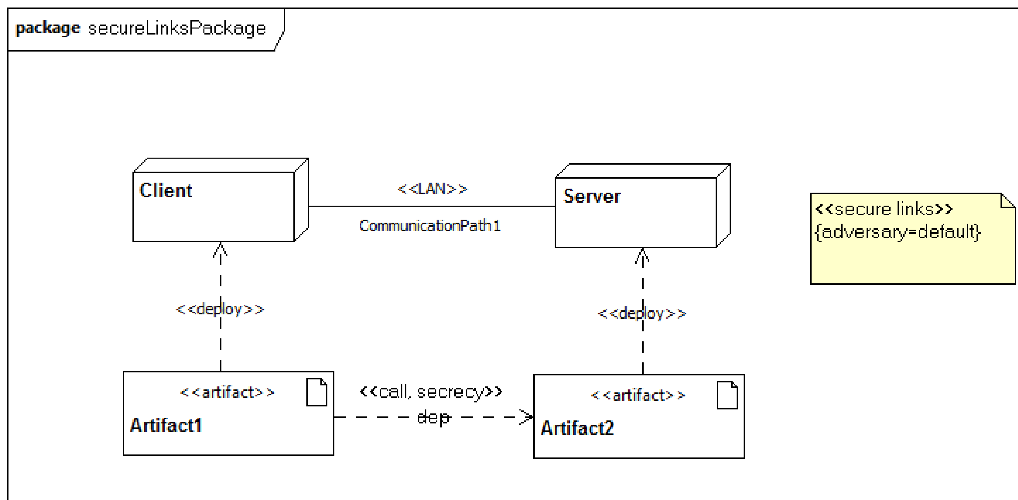


**Abbildung 3.4:** Secure Links [3]

Alle Kommunikationstypen lassen sich durch Stereotype

$$s \in \{<< Internet >>,\ << encrypted >>,\ << LAN >>\}$$

darstellen, wobei für einen Angreifer $A$ und das Stereotyp $s$ gilt

$$Threats_A(s) \subseteq \{delete, read, insert\}.$$

Der Standard-Angreifer default kann folgende Aktionen für Stereotype s durchführen:

$$\begin{aligned} &Threats_{default}(Internet) = \{delete, read, insert\} \\ &Threats_{default}(encrypted) = \{delete\} \\ &Threats_{default}(LAN) = \{\} \end{aligned}$$

Für jede Abhängigkeit $d$ mit Stereotyp t

$$t \in \{<< secrecy >>, << integrity >>\}$$

zwischen zwei verschiedenen, miteinander verbundenen, Kommunikationspunkten muss es einen Pfad geben, so dass

$$\begin{aligned} &wenn\ t == << secrecy >> : read \notin Threats_A(s) \\ &wenn\ t == << integrity >> : insert \notin Threats_A(s) \end{aligned}$$

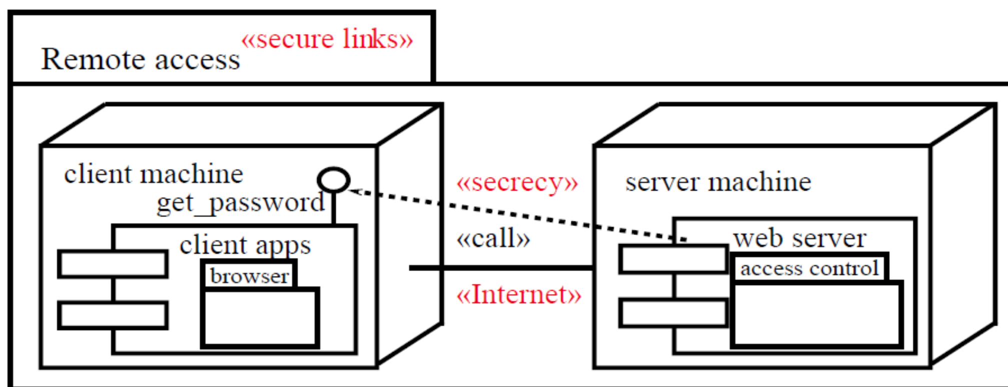


**Abbildung 3.5:** Beispiel <<*secure links*>> [2]

In Abb. 3.5 ist der Hintergrund des oben genannten Käufer-Verkäufer-Szenarios dargestellt: Der Käufer greift mit seinem Gerät über das Internet auf einen Server des Verkäufers zu. Die Verbindung erfolgt gemäß Stereotyp <<*Internet*>>, folglich unverschlüsselt. Der Browser des Käufers äußert einen <<*call*>>, die Antwort des Servers ist mit dem Stereotyp <<*secrecy*>> versehen, also der Anforderung, dass kein Angreifer den Inhalt lesen darf. Folglich kommt es zu einer Verletzung von <<*secure links*>>, da

$$read \in Threats_{default}(Internet).$$

### 3.1.2 BPMN Modell: BPMN2 OCL Check

Der BPMN2 OCL Check ermöglicht es, Anforderungen, die an die Rollen der Beteiligten gestellt werden, zu überprüfen.
Das folgende Diagramm zeigt einen einfachen Ablauf innerhalb der Struktur einer Bank, wobei der *Trader* eine Anfrage stellt und der *Supervisor* diese überprüft:

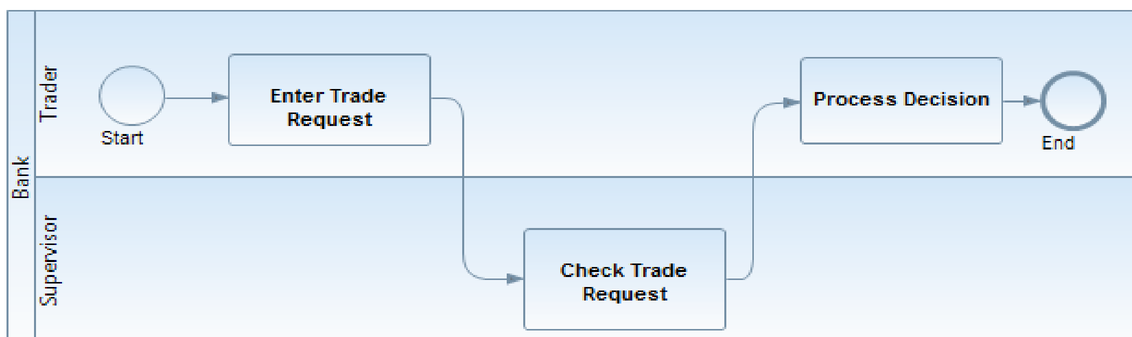


**Abbildung 3.6:** Beispielhafter Handelsablauf aus [3]

Im Diagramm lassen sich drei Rollen erkennen: Die Bank ist der Container für Trader und Supervisor. Der Trader initiiert den Prozess, in dem er die Aufgabe *Enter Trade Request* ausführt. Anschließend wird die Lane vom Trader zum Supervisor gewechselt und die Aufgabe *Check Trade Request* wird ausgeführt. Anschließend erfolgt ein

Wechsel der Lane zum Trader, wo die Aufgabe *Process Decision* ausgeführt und der Container verlassen wird. Damit endet der Prozess.
Ziel ist eine Aufgabentrennung je nach ausübende Rolle - gemäß des *Seperation of Duty* Prinzips. Das bedeutet, dass der Trader die Anfrage, die er gestellt hat nicht durch einen Wechsel zur Lane Supervisor selber überprüfen kann, sondern dies von einem anderen Mechanismus getätigt werden muss. Um es nach dem Beispiel aus 3.2 zu verdeutlichen: Der Käufer sollte nicht selber die Möglichkeit haben, die Zahlung beim Verkäufer auf 'Eingegangen' zu setzen - da er sonst ohne eine Zahlung zu tätigen diese auf den entsprechenden Zustand setzen könnte. Der Verkäufer würde folglich die Ware versenden, ohne dass eine Zahlung eingegangen ist. Klar ist, die Rolle des Überprüfenden übernimmt der Verkäufer.
Diese unerwünschte Rollenteilung lässt sich durch Modifizieren der Abbildung 3.6 leicht verdeutlichen.

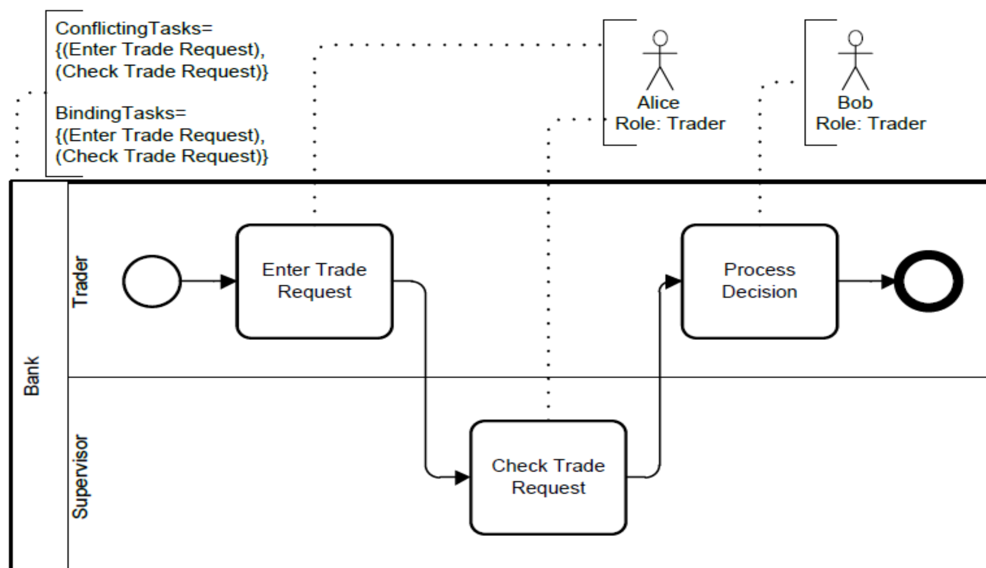


**Abbildung 3.7:** Ungültiges Modell aus [1]

Als Kommentar wurde verdeutlicht, dass die in Konflikt stehenden Aufgaben *Enter Trade Request* und *Check Trade Request* sind. Diese beiden Aufgaben werden jedoch von Alice ausgeführt, was zu einem ungültigen Modell führt.

## 3.2 Funktionalität

CARiSMA basiert auf Eclipse und muss deshalb als zusätzliche Software innerhalb von Eclipse geladen werden. Möchte man beispielsweise UML2 oder BPMN Diagramme analysieren, so ist das entsprechende Repository vorausgesetzt. Ist die Konfiguration erfolgt, muss über das Register 'New' ein neues Analyse-File erstellt werden. Den Wizard 'Analysis' (1) findet man im Ordner CARiSMA. Als nächstes ist ein Projektverzeichnis und ein Name für die Analyse auszuwählen. Im folgenden Schritt lässt sich das zugrunde liegende Modelltyp, beispielsweise BPMN, definieren und das zugehörige Model-File auswählen. Nach erfolgreicher Bestätigung öffnet sich das 'Analysis Editor'-Fenster (2), in dem die durchzuführenden Checks über den Button 'Add' hinzugefügt werden können. Per 'Run'-Button lässt sich die Analyse starten, wobei das Ergebnis im 'Analysis Results'-Fenster (3) zu finden ist.

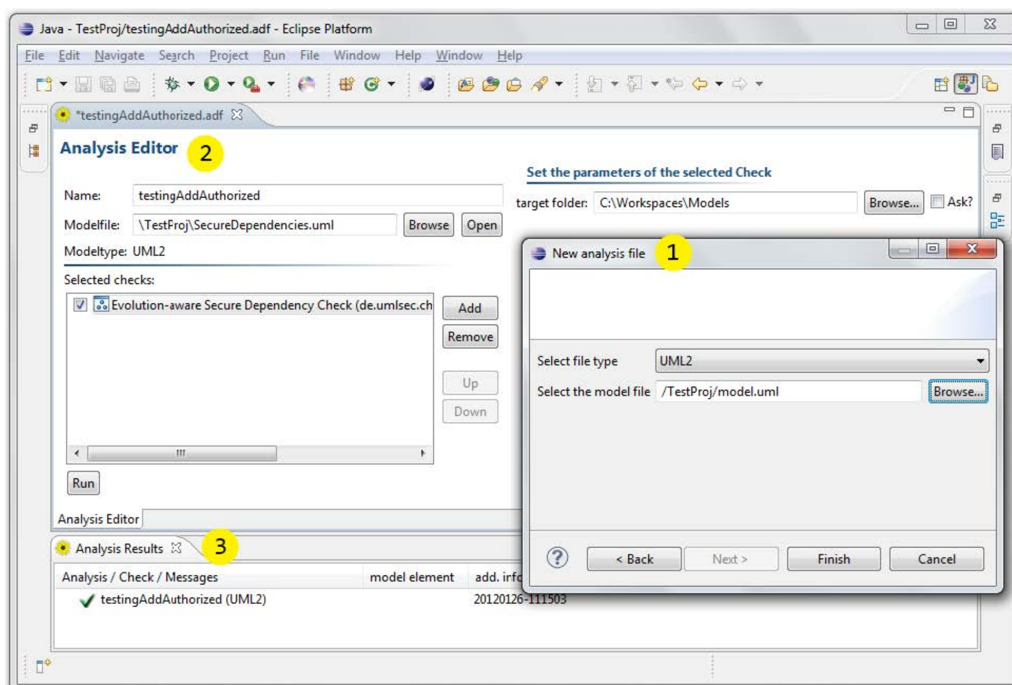


**Abbildung 3.8:** Oberfläche CARiSMA

## 3.3 Komponentenüberblick

Für das Verständnis der Arbeitsweise von CARiSMA werden die einzelnen Komponenten betrachtet. Diese lassen sich in vier Gruppen einteilen: *Core Components, Plugin Definitions, Model Management* und *Analysis Definition and Results*. Der Zusammenhang der Komponenten ist in einem Klassendiagramm in Abb. 3.9 dargestellt. *UMLsec* bildet das zentrale Element der *Core Components* (grün gefärbt), alle anderen Teile dieser Menge können ohne dieses zentrale Element nicht existieren. Die Gruppe der *Plugin Definitions* (rot gefärbt) und *Analysis Definitions and Results* (blau gefärbt) werden vom *Analyzer* genutzt, um eine *Analysis* auszuführen mit anschließender Erstellung eines *AnalysisResult*, wobei dafür notwendigerweise das *AnalysisPlugin* ausgeführt werden muss. Dieses analysiert das zugrunde liegende Modell. Dieses wird vom *ModelLoader* aus der Gruppe *Model Management* (gelb gefärbt) geladen.

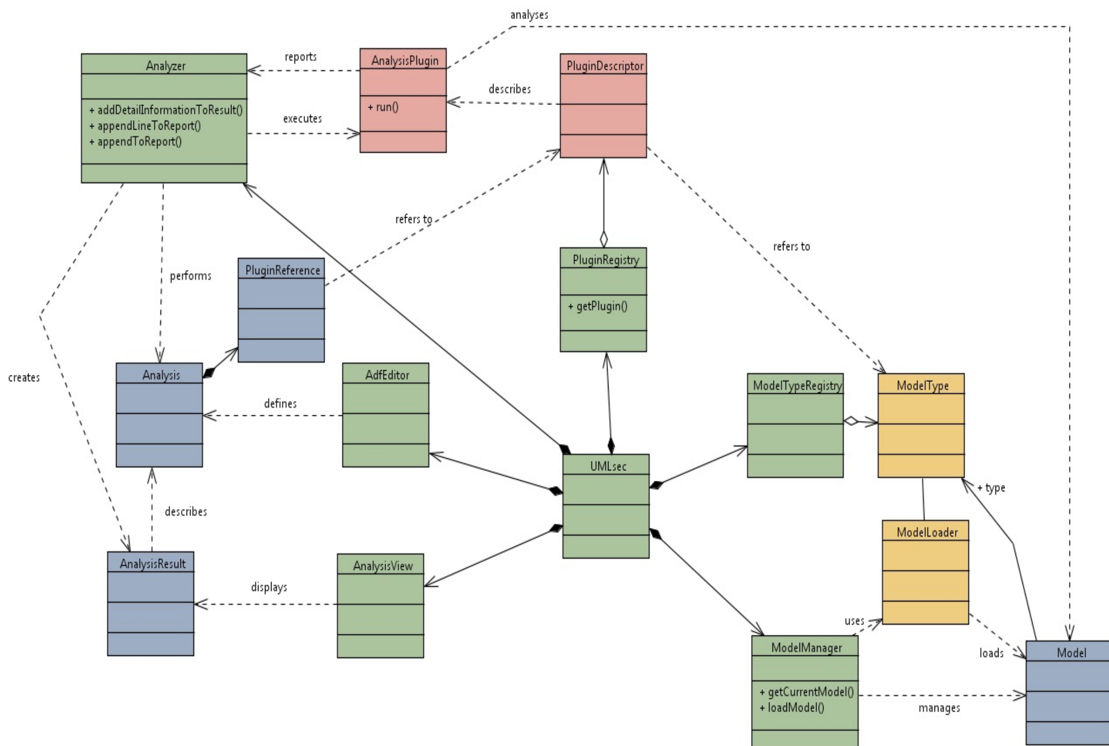


**Abbildung 3.9:** Überblick über die Komponenten aus [10]

# 4 CARiSMA: Analyse eines Beispiels

Im folgenden wird das Beispiel aus Abb. 3.3 noch mal aufgegriffen:

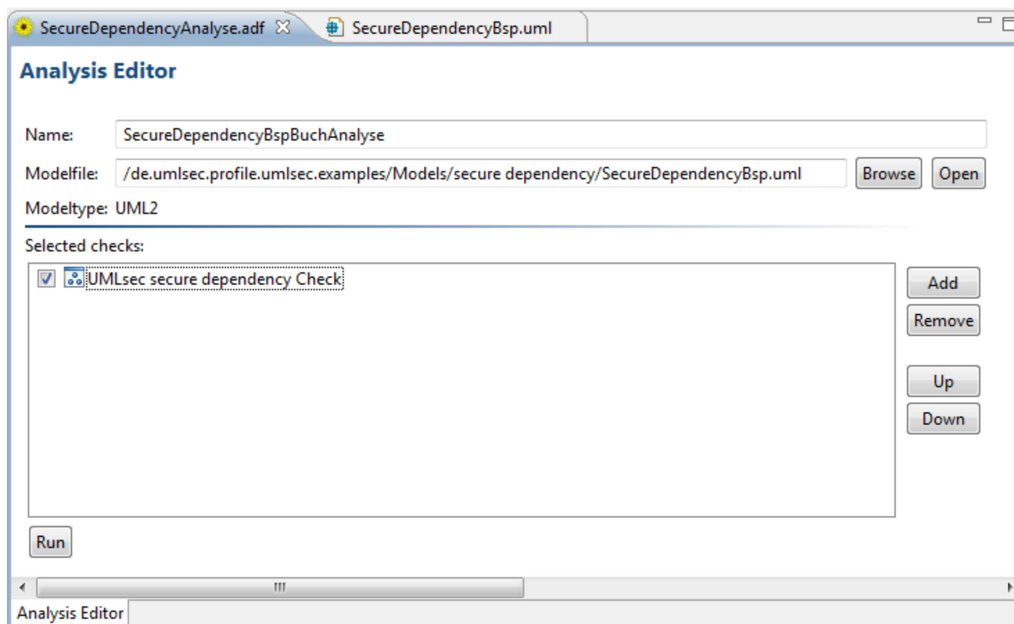


**Abbildung 4.1:** Analysis Editor

Nach Anleitung in Kapitel 3.2 wird ein Analysefile erstellt und das in 3.3 abgebildete Modell ausgewählt. Durch den Button 'Run' wird die Analyse gestartet, das Ergebnis sieht wie folgt aus:

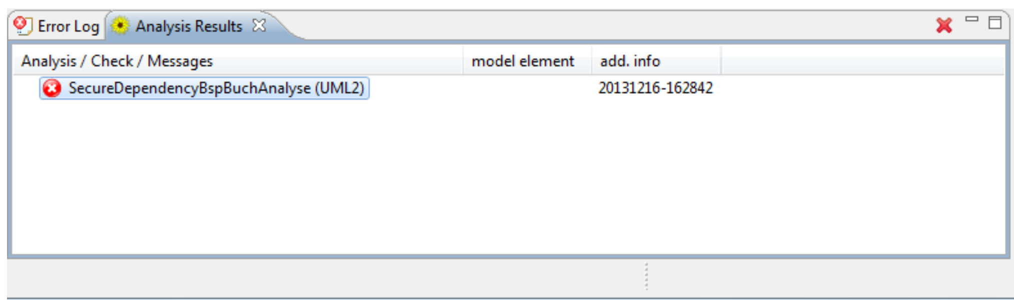


**Abbildung 4.2:** Analysis Result

Im Log findet sich eine Beschreibung des Konflikts. In Kapitel 3 wurde dieser bereits hervorgehoben: Der Key Generator stellt eine Sicherheitsanforderung an newKey()

und random(), wobei der Random generator die Funktion random() bereitstellt und keine Sicherheitsanforderungen an diese stellt. Um nun diesen Missstand zu korrigeren, muss Random generator modifiziert werden:

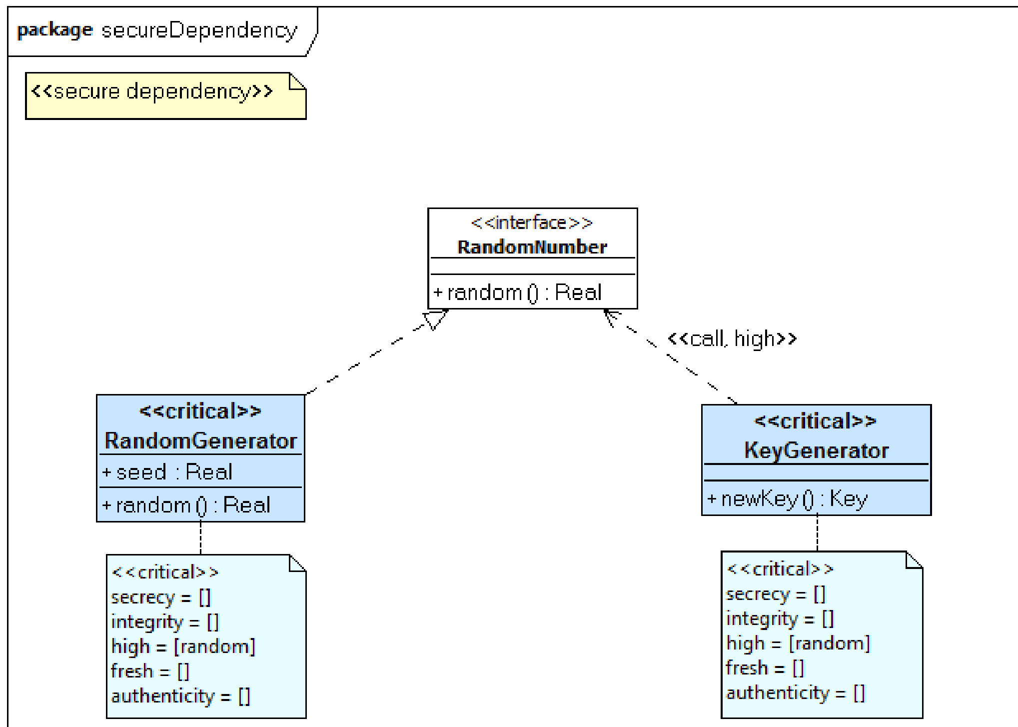


**Abbildung 4.3:** Beispiel eines erfüllten Secure Dependency Modells [3]

Durch das Ergänzen der sicherheitsrelevanten Informationen ist das zugrundeliegende Szenario als Sicher einzustufen, CARiSMA liefert ein erfülltes Ergebnis:

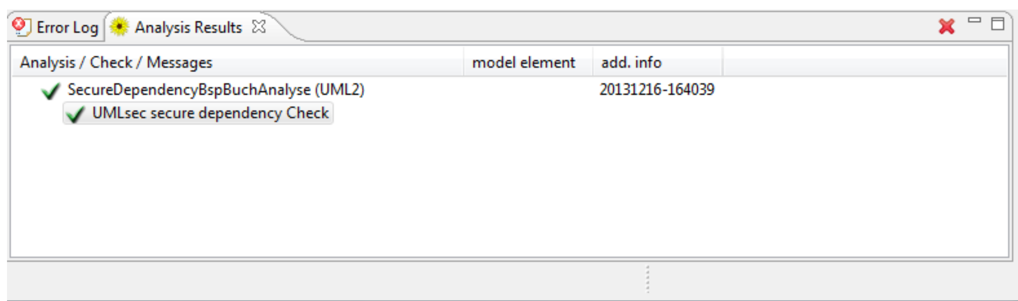


**Abbildung 4.4:** Analysis Result

# 5 Fazit

## 5.1 Zusammenfassung

In dieser Ausarbeitung wurde das CARiSMA-Tool vorgestellt. Zu Beginn wurde erörtert, warum eine Sicherheitsanalyse notwendig ist. Nachfolgend wurden die nötigen Grundlagen aufgebaut, indem eine Einführung in *Unified Modeling Language* und die Erweiterung UMLsec gegeben wurde. Weiterhin wurde die *Object Constraint Language* und die *Business Process Model and Notation* erörtert, wobei die Basiselemente von BPMN2.0 vorgestellt wurden. Da CARiSMA eine Weiterentwicklung des UMLsecTools ist, wurde der Wandel vom UMLsecTool zu CARiSMA hin erörtert, samt der Beweggründe und der neuen Features. Nachfolgend wurden die für CARiSMA elementaren Security Checks vorgestellt, die für die Überprüfung des zugrundeliegenden Modells von Bedeutung sind. Schlussendlich wurde eine Einführung in dem Umgang mit dem Tool geben, konkret ging es um das Erstellen einer Analyse inklusive dem Einbinden des Modells und das Auswählen der Security Checks.

## 5.2 Ausblick

Das CARiSMA-Tool ist aktuell nicht sehr weit verbreitet. Jedoch steigt die Zahl der UML und BPMN Nutzer, da diese beispielsweise in Hochschulen vermehrt gelehrt werden. Zudem gibt es bereits größere Online-Communitys die die Thematik behandeln. Dadurch, dass Sicherheit innerhalb von Geschäftsprozessen und auch Software immer weiter an Bedeutung gewinnt, wird auch die Bedeutung eines Analysetools zunehmen. Grundsätzlich ist es sinnvoll, ein Modell mit Sicherheitsanforderungen wie beispielsweise der UMLsec zu erweitern und durch entsprechende Tools wie CARiSMA zu überprüfen. Diese Art von Testen garantiert jedoch nicht eine Vermeidung von Schwachstellen, viel mehr ist es ergänzender und zusätzlicher Testdurchlauf zu sehen.

# Literatur

[1] Marcel Michel *Bachelorarbeit Konzeption und Umsetzung eines UMLsecTool-Plugins zur Prüfung von Authorization Constraints für die Prozessmodellierungssprache BPMN 2.0* 18. November 2011.

[2] Jan Jürjens *Secure Systems Development with UML* 12. Juli 2004:

[3] Dev Team *CARiSMA Dokumentation http://vm4a003.itmc.tu-dortmund.de/carisma/web/doku.php* Stand 19.11.2013

[4] Object Management Group OCL *Documents associated with Object Constraint Language, Version 2.3.1 http://www.omg.org/spec/OCL/2.3.1/* Stand Januar, 2012

[5] Grafiken, teilweise modifiziert, aus Wikipedia Artikel zu BPMN 2.0 *http://de.wikipedia.org/wiki/Business_Process_Model_and_Notation* Stand 16.12.2013

[6] Object Management Group UML *Unified Modeling Language http://www.uml.org/* Stand 16.12.2013

[7] Object Management Group BPMN *Business Process Model and Notation http://www.omg.org/spec/BPMN/2.0/* Stand 16.12.2013

[8] TU Dortmund, Fakultät für Informatik, Lehrstuhl 14 *http://ls14-www.cs.tu-dortmund.de/ls14/pages/home/index_de.shtml* Stand 16.12.2013

[9] Eclipse Modeling Framework *http://www.eclipse.org/modeling/emf/* Stand 16.12.20131

[10] Sven Wenzel *The UMLsec2 Tool - Development Guide - Interne Entwicklerdokumentation zum UMLsecTool* Stand 16.12.2013