Methodische Entwicklung sicherer CORBA-Anwendungen

Jan Jürjens

Computing Laboratory, University of Oxford und
Software & Systems Engineering, Fak. f. Informatik, TU München
http://www.jurjens.de/jan – juerjens@in.tum.de

Abstract

Das Entwickeln sicherer Software Systeme ist schwierig und fehleranfällig. In vielen implementierten Systemen wurden in der Vergangenheit Verwundbarkeiten gefunden; ein kürzliches Beispiel ist der unauthorisierte Zugang zu Millionen von über das Internet zugänglichen Kontendaten bei einer Amerikanischen Bank.
Wir stellen einen Ansatz zur Lösung dieses Problemes im Kontext der Entwicklung von CORBA-Anwendungen vor. Während die CORBAsecurity Spezifikation zur Common Object Request Broker Architecture (CORBA) der Object Management Group (OMG) einen signifikaten Schritt dahin bietet, objekt-orientierte Technologie für Geschäftsanwendungen brauchbar zu machen, ist sie nicht einfach anzuwenden, insbesondere wenn das security interface eines Object Request Brokers (ORB) benutzt wird, um eine eigene Zugangskontrollpolitik durchzusetzen. Wir zeigen, wie man die Unified Modeling Language (UML), den de-facto Industriestandard in objekt-orientiertem Modellieren, zusammen mit einer formalen Fundierung benutzen kann, um CORBAsecurity Konzepte korrekt anzuwenden.

1 Einleitung

Die heutige Notwendigkeit in vielen Computersystemen, Informationssicherheitsaspekte zu berücksichtigen, ist nicht immer durch ausreichendes Wissen auf Seiten der Entwickler gedeckt. Dies ist problematisch, da in der Praxis Computersicherheit am häufigsten gebrochen wird, indem nicht die Sicherheitsmechanismen selber (wie z.B. Verschlüsselung) direkt angegriffen werden, sondern indem Schwachstellen in der Art und Weise, in der diese *benutzt* werden, ausgenutzt werden [And01]. Demnach können Sicherheitsmechanismen nicht "blind" in einem sicherheitskritischen System eingesetzt werden, sondern die Systementwicklung muss Sicherheitsaspekte von Beginn an berücksichtigen.

Insbesondere Mechanismen für Zugangskontrolle, wie z.B. in der CORBAsecurity Spezifikation [OMG01] sind nicht einfach anzuwenden, insbesondere wenn das security interface eines Object Request Brokers (ORB) benutzt wird, um durch selbstdefinierte access decision objects eine eigene Zugangskontrollpolitik durchzusetzen. Wenn hier ein Fehler passiert, kann die Sicherheit des ganzen Systemes gefährdet sein. Außerdem kann Zugang indirekt und unbeabsichtigt gewährt werden, indem an anderer Stelle im System ein Security Token ausgegeben wird, das zum Zugang berechtigt.

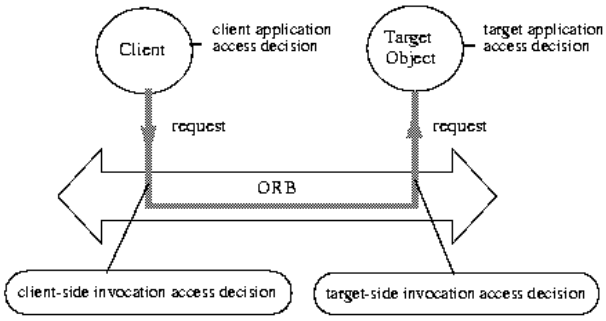

Figure 1: CORBA Zugangsmodell (aus [OMG01])

Diese Arbeit hat zum Ziel, diese Probleme anzugehen, indem eine Methode zur korrekten Entwicklung von sicherheitskritischen CORBA Anwendungen, die von den CORBAsecurity Spezifikationen, und insondere der CORBAsecurity Zugangskontrolle Gebrauch machen, bereitgestellt wird. Hierfür verwenden wir einen Teil der weit gebräuchlichen objekt-orientierten Design-Sprache Unified Modeling Language (UML) [RJB99], um die sicherheitskritische Anwendung zu spezifizieren, und zu überprüfen, ob die gestellten Sicherheitsanforderungen erfüllt sind.

Übersicht Nach Hintergrundinformationen über die CORBA Sicherheitsspezifikation und insbesondere den enthaltenen Zugangskontrollmechanismus, erläutern wir unseren Gebrauch der UML in Abschnitt 3. In Abschnitt 4 skizzieren wir einen Teil eines Designprozesses mit dem Ziel, Zugangskontrollregeln in CORBA durchzusetzen. In Abschnitt 5 illustrieren wir unseren Zugang mit dem Beispiel einer web-basierten Finanzanwendung. Wir schließen mit einem Überblick über verwandte Arbeiten und einem Fazit. Dieser Extended Abstract enthält nur einen Überblick, weitere Einzelheiten werden in der vollständigen Version gegeben.

2 Zugangskontrolle in CORBA

In CORBA steuert die Zugriffssteuerungspolitik für Objektaufrufe (object invocation access policy) den Zugriff eines Client auf ein bestimmtes Objekt über eine bestimmte Methode. Dieses wird vom ORB und dem Security Service realisiert [Bla99] (s. Abbildung 1).

Nur wenn eine bestimmte Operation von der Zugriffssteuerungspolitik genehmigt wird, kann der Client sie auf dem Target-Objekt aufrufen. Hierzu werden Zugriffsentscheidungsfunktionen (access decision functions) verwendet, die auf die Frage, ob der Zugriff erlaubt ist, eine positive oder negative Antwort liefern. Diese Entscheidung hängt ab von

- der aufgerufenen Operation,
- den Privilegien des Principals, in dessen Auftrag der Client handelt
- und Kontrollattributen (control attributes) des Target-Objekts.

Auf diese Weise ist es möglich, Zugangskontrolle zu Objekten über *protection domains* statt auf der Ebene einzelner Objekte zu regeln, da letzteres in Systemen realistischer Größe kaum machbar wäre.

Beim Starten eines Object Request Brokers (ORB) wird ein neuer Ausführungskontext (*execution context*) geschaffen, der u.a. *credentials* enthählt (sicherheitskritische Informationen wie etwa Identität und Privilegien des Aufrufenden des ORB). Diese können folgender Art sein:

- *eigene* credentials,
- *erhaltene* credentials von aufrufenden Objekten, oder
- *aufrufende* credentials, die an aufgerufene Objekte geschickt werden (im Fall von *Delegation* können diese von den eigenen credentials verschieden sein).

Der aktuelle Ausführungskontext eines Programmes ist in dem **Current** Objekt enthalten, das durch den ORB gelesen werden kann.

Die Zugangskontrolle wird vom ORB in Abhängigkeit vom aktuellen Ausführungskontext und dem protection domain eines Objektes durchgesetzt. Hierbei kann das security interface eines Object Request Brokers (ORB) benutzt werden, um eine eigene Zugangskontrollpolitik durchzusetzen.

In CORBA können jeder Methode bis zu vier verschiedene Rechte zugewiesen werden, die beim aufrufenden Objekt vorhanden sein müssen, um Zugang zu der Methode zu erhalten:

g Das "get" Recht kontrolliert Zugang zu Methoden, die Informationen an das aufrufende Objekt zurückgeben.

s Das "set" Recht kontrolliert Zugang zu Methoden, die Informationen im aufgerufenen Objekt ändern.

u Das "use" Recht kontrolliert Zugang zu Methoden, die das aufgerufene Objekt eine Aufgabe ausführen lassen.

m Das "manage" Recht kontrolliert Zugang zu Methoden, die nicht normalen Benutzern (sondern z.B. System Administratoren) zugänglich sein sollen.

CORBAsecurity bietet weiterhin den Schutz von Nachrichten bzgl. Authentifikation, Vertraulichkeit und Integrität, die mit kryptographischen Mitteln gewährleistet werden (Authentifikation und Integrität durch Signieren, Vertraulichkeit durch Verschlüsseln).

Die Zugangskontrollentscheidungen werden von *access decision objects* (ADO's) getroffen. Diese können insbesondere auch frei auf der Anwendungsebene definiert werden [OMG01, 2-50], siehe in dem Beispiel in Abschnitt 5.

3 Entwickeln sicherer CORBA Anwendungen mit UML

Wir benutzen einen Teil der Unified Modeling Language (UML) [UML01], dem de-facto Industriestandard in objekt-orientiertem Modellieren, zusammen mit einer formalen Semantik. Diese benutzen wir, um sicherzustellen, dass die eingesetzten Schutzmechanismen die Sicherheitsanforderungen erfüllen. Genauer überprüfen wir anhand der formalen Semantik, dass das spezifizierte dynamische Verhalten die Sicherheitspolitiken erfüllt. Sicherheitsanforderungen auf der Spezifikationsebene (im Gegensatz zur Implementationsebene) zu überprüfen, hat den Vorteil, dass Designfehler so früh wie möglich erkannt und behoben werden können, was zu erheblichen Einsparungen in der Entwicklung

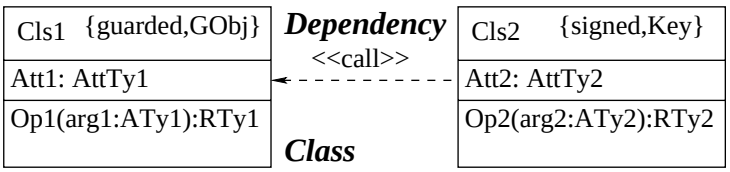

Figure 2: Klassendiagramm

und Wartung von Systemen führt. Außerdem ist eine formale Analyse auf abstrakterer Ebene einfacher durchzuführen (auch wenn Programmierfehler dadurch natürlich nicht aufgedeckt werden können).

UML besteht aus verschiedenen Arten von Diagrammen, die verschiedene Ansichten eines Systemes wiedergeben. Um die Darstellung zu vereinfachen, benutzen wir hier nur einen Teil von UML. Weiter verwenden wir die Standard-Erweiterungsmechanismen von UML (Stereotypes, Tags und Constraints), um Sicherheitsaspekte zu formulieren.

Wir benutzen die folgenden Diagrammarten

Klassendiagramme definieren die statische Struktur des Systems: Klassen mit Attributen und Operationen, und Beziehungen zwischen Klassen.

Zustandsdiagramme (Statechart diagrams) geben das dynamische Verhalten eines individuellen Objektes: Ereignisse können Zustandsänderungen oder Aktionen hervorrufen.

Einsatzdiagramme (Deployment diagrams) beschreiben die physische Ebene des Systemes und seiner Umgebung.

Wir definieren die Diagramme mithilfe ihrer abstrakten Syntax, um Verifikation zu ermöglichen. Wir geben auch die konkrete Syntax, da wir sie in dem Fallbeispiel aus Gründen der Lesbarkeit verwenden.

3.1 Klassendiagramme

Eine *Attributespezifikation* $A = (\mathsf{att\_name}, \mathsf{att\_type})$ ist gegeben durch einen Namen $\mathsf{att\_name}$ und einen Datentyp $\mathsf{att\_type}$. Eine *Operationsspezifikation* $O = (\mathsf{op\_name}, \mathsf{Arguments}, \mathsf{op\_type})$ ist gegeben durch einen Namen $\mathsf{op\_name}$, eine Menge $\mathsf{Arguments}$ von Argumenten und den Datentyp $\mathsf{op\_type}$ des Rückgabewertes. Die Argumentenmenge kann leer sein; der Rückgabewert $\emptyset$ bedeutet Abwesenheit eines Rückgabewertes. Ein *Argument* $A = (\mathsf{arg\_name}, \mathsf{arg\_type})$ wird gegeben durch seinen Namen $\mathsf{arg\_name}$ und seinen Datentyp $\mathsf{arg\_type}$. Ein *Klassenmodell* $C = (\mathsf{class\_name}, \mathsf{AttSpecs}, \mathsf{OpSpecs}, \mathsf{State})$ ist gegeben durch einen Namen $\mathsf{class\_name}$, eine Attributmenge $\mathsf{AttSpecs}$, eine Menge $\mathsf{OpSpecs}$ von Operationsspezifikationen und ein Zustandsdiagramm State, das das Objektverhalten spezifiziert. Ein *Klassendiagramm* $D = (\mathsf{Cls}, \mathsf{Dependencies})$ ist gegeben durch eine Menge Cls von Klassenmodellen und eine Menge $\mathsf{Dependencies}$ von Abhängigkeiten. Eine *Abhängigkeit* ist ein Tupel $(\mathsf{client}, \mathsf{supplier}, \mathsf{stereotype})$ bestehend aus den Klassennamen client und $\mathsf{supplier}$ und einem Label $\mathsf{stereotype}$ (genannt Stereotyp), das die Art der Abhängigkeit angibt (zum Bespiel «*call*»).

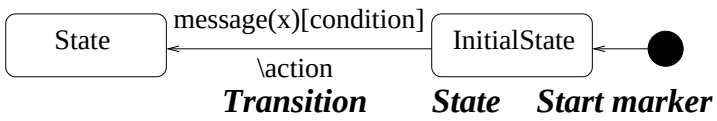

Figure 3: Zustandsdiagramm

3.2 Zustandsdiagramme

Wir benutzen Zustandsdiagramme, um das Verhalten von Objekten zu spezifizieren, insbesondere der Access Decision Objects.

Wir definieren Zustandsdiagramme: Ein *Zustandsdiagramm* $S = (\mathsf{States}, \mathsf{init\_state}, \mathsf{Transitions})$ ist gegeben durch eine Menge States von Zuständen (inklusive des Anfangszustandes init_state) und einer Menge Transitions von Transitionen. (In der konkreten Syntax, der Anfangszustand ist durch einen runden Punkt markiert.) Eine *Transition* $t = (\mathsf{source}, \mathsf{event}, \mathsf{condition}, \mathsf{Actions}, \mathsf{target})$ hat einen Anfangszustand source, ein triggerndes Ereignes event, eine logische Bedingung guard, eine Liste Actions von Aktionen und einen Zielzustand target. Ein Ereignis *event* ist der Name einer Operation mit einer Lister von Variablen als Argumente (zum Beispiel $\mathsf{op}(x, y, z)$). Eine *Aktion* kann entweder sein, einen Wert v einem Attribut a zuzuweisen (geschrieben als $a := v$), oder eine Operation op mit Werten $v_1, \ldots, v_n$ aufzurufen (geschrieben als $\mathsf{op}(v_1, \ldots, v_n)$, oder einen Rückgabewert v als Antwort auf einen früheren Aufruf der Operation op zurückzugeben (geschrieben als $\mathsf{return}_{\mathsf{op}}(v)$). In der konkreten Syntax stehen Bedingungen in eckigen Klammern, und Aktionen ist ein Schrägstrich vorangestellt.

3.3 Einsatzdiagramme

Einsatzdiagramme beschreiben die physische Ebene eines Systems, mit den Aufenthaltsorten der verschiedenen Komponenten des Systems und Informationen über die Art der verwendeten Kommunikationsverbindungen zwischen verschiedenen Komponenten, die Bedrohungsszenarien induzieren.

Ein *Knoten* $N = (\mathsf{location}, \mathsf{Components})$ ist gegeben durch seinen Ort location (zum Beispiel die URL oder "local system" für ein lokales System) und eine Menge Components der enthaltenen Komponenten. Ein *Einsatzdiagramm* $D = (\mathsf{Nodes}, \mathsf{Links}, \mathsf{Dependencies})$ ist gegeben durch eine Menge Nodes von Knoten, eine Menge Links von Kommunikationsverbindungen zwischen Knoten und einer Menge Dependencies von Abhängigkeiten zwischen Komponenten. Eine *Verbindung* $l = (\mathsf{nds}, \mathsf{stereo})$ besteht aus einer zweielementigen Menge nds von verbundenen Knoten und einem Stereotyp der die Art der Verbindung angibt (zum Beispiel «*Internet*»). Eine *Abhängigkeit* bezeichnet hier ein Tupel $(\mathsf{client}, \mathsf{supplier}, \mathsf{interface}, \mathsf{tag})$ bestehend aus den Komponenten client und supplier.

4 Design Prozess

Wir skizzieren einen Teil eines Design Prozesses für sichere CORBA Anwendungen mit UML, mit Schwerpunkt auf der CORBA Zugangskontrolle.

(1) Zugangsregeln für den Zugang zu sicherheitskritischen Objekten formulieren.

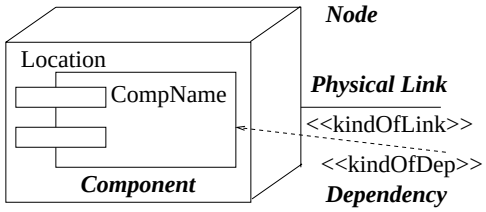

Figure 4: Deployment diagram

(2) Überprüfen der Zugangsregeln im Security Service mit Zustandsdiagrammen spezifizieren.

(3) Feststellen, ob die Objekte durch den Security Service ausreichend geschützt werden durch Zeigen, dass nur Zugang gewährt wird, der von den Sicherheitsanforderungen zugelassen wird.

(4) Feststellen, dass die Zugangskontrolle konsistent mit der vom System zu erbringenden Funktionalität ist durch Zeigen, dass die von geschützten Objekten *abhängigen* Objekte das erwartete Verhalten ausführen können.

Somit müssen wir die folgenden beiden Anforderungen überprüfen:

Sicherheitsanforderung: Zugangskontrolle ist restriktiv genug (unter Einbezugnahme des in den Einsatzdiagrammen spezifizierten Bedrohungsszenarios).

Funktionalitätsanforderung: Zugangskontrolle ist nicht zu restriktiv (d.h. verwehrt keinen legitimen Zugang).

Die Funktionalitätsanforderung ist wichtig, um die Anforderung der Verfügbarkeit von Anfang an zu berücksichtigen: Es nicht immer klar ist, ob Sicherheitsanforderungen tatsächlich implementierbar sind (und sich nicht etwa widersprechen). Wird ein diesbezügliches Problem erst während der Implementierungsphase festgestellt, führt dies zum einen zu erhöhten Kosten, da das Design korrigiert werden muss, zum anderen wird vermutlich der Sicherheitsstandard erniedrigt, um die Kostensteigerung zu begrenzen.

5 Beispiel: Finanzanwendung

Wir illustrieren unsere Methode am Beispiel einer Internet-basierten Finanzanwendung. Obwohl aus Darstellungsgründen relativ klein, ist das Beispiel doch realistisch, insofern es einige typische Punkte im Zusammenhang mit Zugangskontrolle für Internet-basierte E-commerce Anwendungen aufzeigt (nämlich, verschiedene Stellen – Dienstanbieter und Kunden – zu haben, die miteinander über ein unsicheres Netzwerk interagieren und sich gegenseitig ein begrenztes Maß an Vertrauen entgegenbringen).

Wir beschreiben zunächst die physische Ebene der Anwendung in einem UML Einsatzdiagramm und formulieren die Sicherheitsanforderungen. Wir zeigen in UML Diagrammen, wie diese Sicherheitsanforderungen mithilfe der CORBA Zugangskontrolle durchgesetzt werden. Die durch die UML Diagramme gegebene Spezifikation kann dann daraufhin überprüft werden, ob sie sicher ist, indem nachgewiesen wird, dass kein Zugang gewährt wird, der nicht von den Sicherheitsanforderungen impliziert wird.

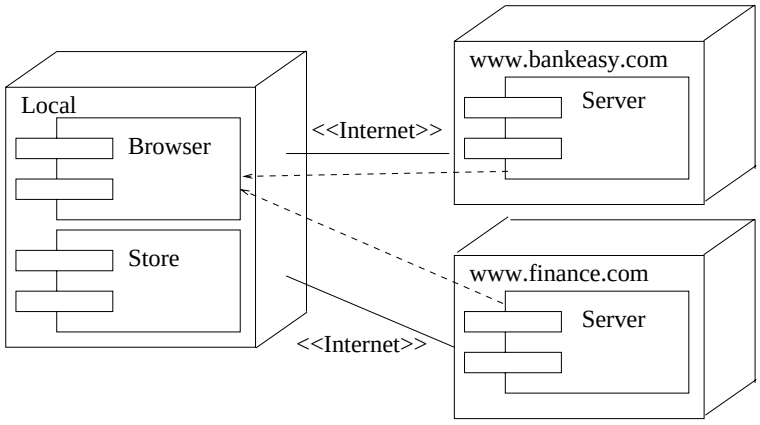

Figure 5: Einsatzdiagramm

Zwei (fiktionale) Institutionen bieten Kunden Dienste über das Internet an: Eine Internet Bank, Bankeasy, und ein Finanzberator, Finance. Die physische Ebene ist also gegeben in Abbildung 5.

Im Rahmen dieser Dienste müssen Objekte dieser Institutionen auf Daten auf dem lokalen System der Benutzer zugreifen. Die Struktur dieses Zugriffs ist verdeutlicht in dem Klassendiagramm in Abbildung 6.

Um dies zu verwirklichen, muss der lokale Benutzer Objekten verschiedener Anbietern bestimmte Privilegien gewähren.

(1) Objekte der Bank können die Finanzdaten in der lokalen Datenbank schreiben und lesen, aber nur zwischen 13 und 14 Uhr (wenn der Kunde das Konto normalerweise betreut).

(2) Objekte (zum Beispiel des Finanzberaters) können einen Auszug der Finanzdaten lesen, der für diesen Zweck erzeugt wird. Da diese Information nur lokal benutzt werden darf, müssen die Objekte von einer Zertifizierungsfirma, CertiFlow, authorisiert werden, die zertifiziert, dass sie keine Informationen durch verdeckte Kanäle aussenden.

(3) Objekte des Finanzberaters dürfen den Micropayment Schlüssel des lokalen Benutzers gebrauchen (um in seinem Auftrag Aktieninformationen zu kaufen), aber dieser Zugang soll nur fünf Mal pro Woche gewährt werden.

Der Zugang zu den lokalen Finanzdaten wird durch den CORBA Security Service geregelt. Abbildung Abbildung 7 zeigt ein UML Subsystem, das den relevanten, stark vereinfachten Teil des CORBA Security Service enthält, zusammen mit den Access Decision Objects der geschützten Objekte (das Verhalten der geschützten Objekte wurde hier fortgelassen).

Die Zugangsregeln werden spezifiert durch die Zustandsdiagramme in Abbildung 7 (wobei wir annehmen, dass die Bedingung slot genau zwischen 13 und 14 Uhr erfüllt ist, dass limit genau dann erfüllt ist, wenn der Zugang zum Micropayment Schlüssel in der aktuellen Woche weniger als fünf gewährt worden ist; wie dies implementiert wird, lassen wir hier aus).

Hier gehen wir davon aus, dass der CORBA Security Service anhand von *credentials*

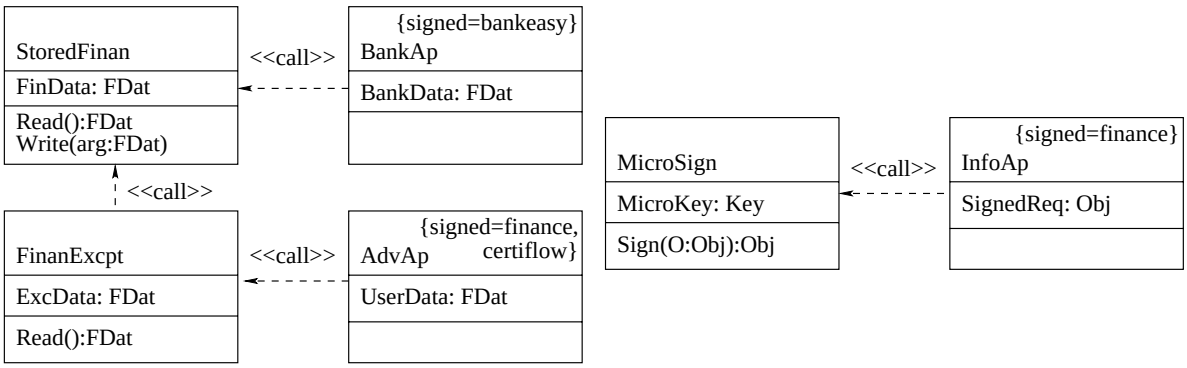

Figure 6: Klassendiagramm

überprüft, dass ein Prinzipal, im Auftrag dessen ein Objekt ausgeführt wird, tatsächlich von der Stelle *sig* authorisiert ist, dessen Name als zweites Argument von getObj gegeben ist. Fehlerbehandlung wird hier zur Vereinfachung nicht berücksichtigt.

Nun können wir entsprechend Schritt (3) in Abschnitt 4 zeigen, dass die UML Spezifikation keinen Zugang erlaubt, der nicht von den Zugangsregeln in (1)–(3) erlaubt wird. Da es sich hier um ein sehr einfaches Beispiel handelt, lässt sich dies anhand der gegebenen UML Zustandsdiagramme zeigen:

- Angenommen, den CORBA Security Service erreicht eine Anforderung des Objektes StoFi. Diese Anforderung wird an das zugehörige Access Decision Object FinGd weitergereicht. Laut dessen Spezifikation wird der Anforderung stattgegeben, falls der anfordernde Prinzipal von der Bank authorisiert ist und falls die Bedingung an den Zeitabschnitt erfüllt ist, wie in der obigen Zugangsregeln erfordert.

- Wenn den CORBA Security Service eine Anforderung des Objektes FinEx erreicht, wird diese an das zugehörige Access Decision Object ExcGd weitergereicht. Der Anforderung wird stattgegeben, falls der anfordernde Prinzipal von der Zertifikationsinstanz CertiFlow authorisiert ist.

- Falls den CORBA Security Service eine Anforderung des Objektes MicSi erreicht, wird diese an das zugehörige Access Decision Object MicGd weitergereicht. Laut dessen Spezifikation wird der Anforderung stattgegeben, falls der anfordernde Prinzipal vom Finanzberater authorisiert ist und falls die Bedingung an die Anzahl der Zugänge erfüllt ist.

Bei komplizierteren Spezifikationen muss die Verifikation anhand einer formalen Semantik für den verwendeten Teil der UML durchgeführt werden (und zwar vorzugsweise in einem werkzeugunterstützten Vorgehen). Aus Platzgründen kann dies hier nicht demonstriert werden; wir verweisen auf [Jür02a, Jür02b].

Hinsichtlich Schritt (4) in Abschnitt 4 kann man nun zeigen, dass die von geschützten Objekten abhängige Objekte in gewünschter Weise Zugang erhalten. Wieder verzichten wir, da es sich um ein einfaches Beispiel handelt, aus Platzgründen auf die Verwendung der formalen Semantik, sondern beziehen uns direkt auf die UML Diagramme.

- Angenommen, der Zugriff auf das Objekt StoFi wird verweigert. Laut Spezifikation es zugehörigen Access Decision Objects FinGd folgt daraus, dass entweder der anfordernde Prinzipal von der Bank nicht authorisiert ist, oder die Bedingung an den Zeitabschnitt nicht erfüllt ist.
- Wenn der CORBA Security Service eine Anforderung des Objektes FinEx abweist, schliessen wir aufgrund der Spezifikation des zugehörigen Access Decision Object ExcGd, dass der anfordernde Prinzipal nicht von der Zertifikationsinstanz CertiFlow authorisiert ist.
- Angenommen, der Zugriff auf das Objektes MicSi wird abgelehnt. Nach Spezifikation des zugehörigen Access Decision Objects MicGd bedeutet dies, dass der anfordernde Prinzipal nicht vom Finanzberater authorisiert ist, oder die Bedingung an die Anzahl der Zugänge nicht erfüllt ist.

6 Literaturüberblick

Die hier präsentierte Arbeit ist Teil eines allgemeineren Ansatzes zum Entwurf sicherer Systeme mit UML: [Jür01a] präsentiert allgemeine Konzepte, [Jür02c] einen Ansatz unter Verwendung von Bedrohungsbäumen. [Jür01b] wendet den Ansatz auf das Beipiel der Common Eletronic Purse Specifications an. Weitere Einzelheiten und Literaturhinweise sind in [Jür02b].

Mit Methoden zur Entwicklung sicherer Systeme befassen sich auch die folgenden drei Beiträge: Bei dem Ansatz in [EM97] werden anwendungsspezifische Sicherheitspolitiken in einer Sicherheitsanforderungslogik spezifiziert. Abstrakte Spezifikationen werden in einem top-down Zugang in einer Programmiersprache implementiert, deren Sprachkonzepte an das Spezifikationsmodell angepasst sind. Unserer Ansatz hier unterscheidet sich von dieser Arbeit dadurch, dass wir als Spezifikationssprache mit der UML eine weit verbreitete Notation verwenden, und sie speziell in dieser vorliegenden Arbeit auf verbreitete sicherheitsrelevante Programmierkonzepte (CORBAsecurity) anwenden.

[HF98] präsentiert einen werkzeugunterstützten Entwicklungsprozess für sichere Chipkartenanwendungen unter Einsatz formaler Methoden. Die verwendete formale Spezifikationssprache ist eine Temporallogik (die CTL von Clarke und Emerson); Spezifikationen können mit dem Model-Checker SMV der Carnegie-Mellon University automatisch verifiziert werden. Wiederum ist der Unterschied, dass CTL in der Industrie nicht denselben Bekanntheitsgrad geniesst wie UML. Vorteil der zitierten Arbeit ist allerdings die dort vorhandene Werkzeugunterstützung, die für unseren Zugang hier noch in Arbeit ist.

[Lot00] schlägt einen methodischen Rahmen zur Entwicklung sicherer Systeme unter Verwendung formaler Methoden vor. Obwohl entwickelt anhand Spezifikationsmethode Focus [BS01], liesse sich dieser Ansatz auch auf die hier verwendete Notation der UML anwenden.

Mit CORBAsecurity befassen sich die folgenden Arbeiten. [ALP+00] diskutiert Sicherheitsschwachstellen in CORBAsecurity. Ein weiterer interessanter Ansatz könnte der Versuch sein, unsere UML Methodik dahingehend zu erweitern, dass diese Schwachstellen bei der Benutzung des CORBA Security Service vom Anwendungsentwickler umgangen werden. [Kar00, BRV02] geben formale Analysen für Teile des CORBA Security Ser-

vice. Diese Ansätze sind insofern komplementär zu unserem Ansatz, als es dort um die Korrektheit des CORBA Security Service geht, und hier um die Korrektheit von Anwendungen, die diesen Service verwenden. [Bro98] präsentiert eine Zugangskontrolle-Spezifikationssprache für CORBA. Es wäre zu überlegen, ob sich diese in unseren Ansatz integrieren liesse.

7 Fazit

Wir haben einen Teil der Unified Modeling Language (UML) verwendet, um Zugangskontrollregeln in CORBA Anwendungen zu spezifizieren und verifizieren.

Die vorgestellte Methode scheint praktisch machbar und nutzbringend:

- Die Anwendung der CORBA Zugangskontrolle ist in der Praxis nicht einfach, insbesondere wenn das security interface eines Object Request Brokers (ORB) benutzt wird, um eine eigene Zugangskontrollpolitik durchzusetzen. Somit scheint eine Unterstützung durch UML Spezifikationen hilfreich.
- In diesem Extended Abstract konnten wir nur ein einfaches Beispiel darstellen. Aufgrund der von UML unterstützten hohen Abstraktionsebene ist aber zu erwarten, dass die Methode sich auf Systeme realistischer Größe anwenden lässt, insbesondere da im Allgemeinen nur ein kleiner Teil des Systems sicherheitskritisch ist.

Laufende Arbeit betrifft die Werkzeugunterstützung für unsere Verwendung von UML für die Entwicklung sicherer CORBA-Anwendungen (unter Verwendung eines Zustandsdiagramm-Simulators).

Weitere Arbeit wird sich der verbleibenden Teile der CORBAsecurity annehmen (z.B. auditing und non-repudiation).

References

[ALP+00] A. Alireza, U. Lang, M. Padelis, R. Schreiner, and M. Schumacher. The Challenges of CORBA Security. In M. Schumacher and R. Steinmetz, editors, *Sicherheit in Netzen und Medienströmen, Tagungsband des GI Workshops "Sicherheit in Mediendaten"*, 2000.

[And01] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems.* Wiley, 2001.

[Bla99] B. Blakley. *CORBA Security.* Addison-Wesley, 1999.

[Bro98] G. Brose. Towards an Access Control Policy Specification Language for CORBA. In *ECOOP 1998 Workshop on Distributed Object Security (EWDOS'98)*, 1998.

[BRV02] David Basin, Frank Rittinger, and Luca Viganò. A Formal Analysis of the CORBA Security Service. In *The 2nd International Z and B Conference*, 2002.

[BS01] M. Broy and K. Stølen. *Specification and Development of Interactive Systems.* Springer-Verlag, 2001.

[EM97] C. Eckert and D. Marek. Developing secure applications: A systematic approach. In *IFIP TC11 13th International Conference on Information Security*, pages 267 – 279. Chapman & Hall, 1997.

[HF98] M. Horak and R. Falk. Entwicklungsprozeß und Werkzeugunterstützung für sichere Chipkartenanwendungen. In K. Bauknecht, A. Büllesbach, H. Pohl, and S. Teufel, editors, *Sicherheit in Informationssystemen (SIS '98)*, pages 117–138. vdf Hochschulverlag, 1998.

[Hor00] P. Horster, editor. *Systemsicherheit*. Vieweg Verlag, 2000. Conference proceedings.

[Jür01a] J. Jürjens. Developing secure systems with UMLsec — from business processes to implementation. In D. Fox, M. Köhntopp, and A. Pfitzmann, editors, *VIS 2001*. Vieweg-Verlag, 2001.

[Jür01b] J. Jürjens. Modelling audit security for smart-card payment schemes with UMLsec. In M. Dupuy and P. Paradinas, editors, *Trusted Information: The New Decade Challenge*, pages 93–108. IFIP, Kluwer Academic Publishers, 2001. Proceedings of SEC 2001 – 16th International Conference on Information Security.

[Jür02a] J. Jürjens. Formal Semantics for Interacting UML subsystems. In *5th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2002)*. IFIP TC6/WG6.1, Kluwer Academic Publishers, 2002.

[Jür02b] J. Jürjens. *Principles for Secure Systems Design*. PhD thesis, Oxford University Computing Laboratory, 2002. Submitted.

[Jür02c] J. Jürjens. Using UMLsec and Goal-Trees for Secure Systems Development. In *Symposium of Applied Computing*. ACM, 2002.

[Kar00] G. Karjoth. Authorization in CORBA security. *Journal of Computer Security*, 8(2,3):89–108, 2000.

[Lot00] V. Lotz. Ein methodischer Rahmen zur formalen Entwicklung sicherer Systeme. In *[Hor00]*, 2000.

[OMG01] OMG. Corbaservices: Security service specification v.1.7, 8 March 2001. Available at http://www.omg.org/cgi-bin/doc?formal/2001-03-08.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[UML01] UML Revision Task Force. OMG UML Specification v. 1.4. OMG Document ad/01-09-67. Available at http : //www.omg.org/uml, September 2001.

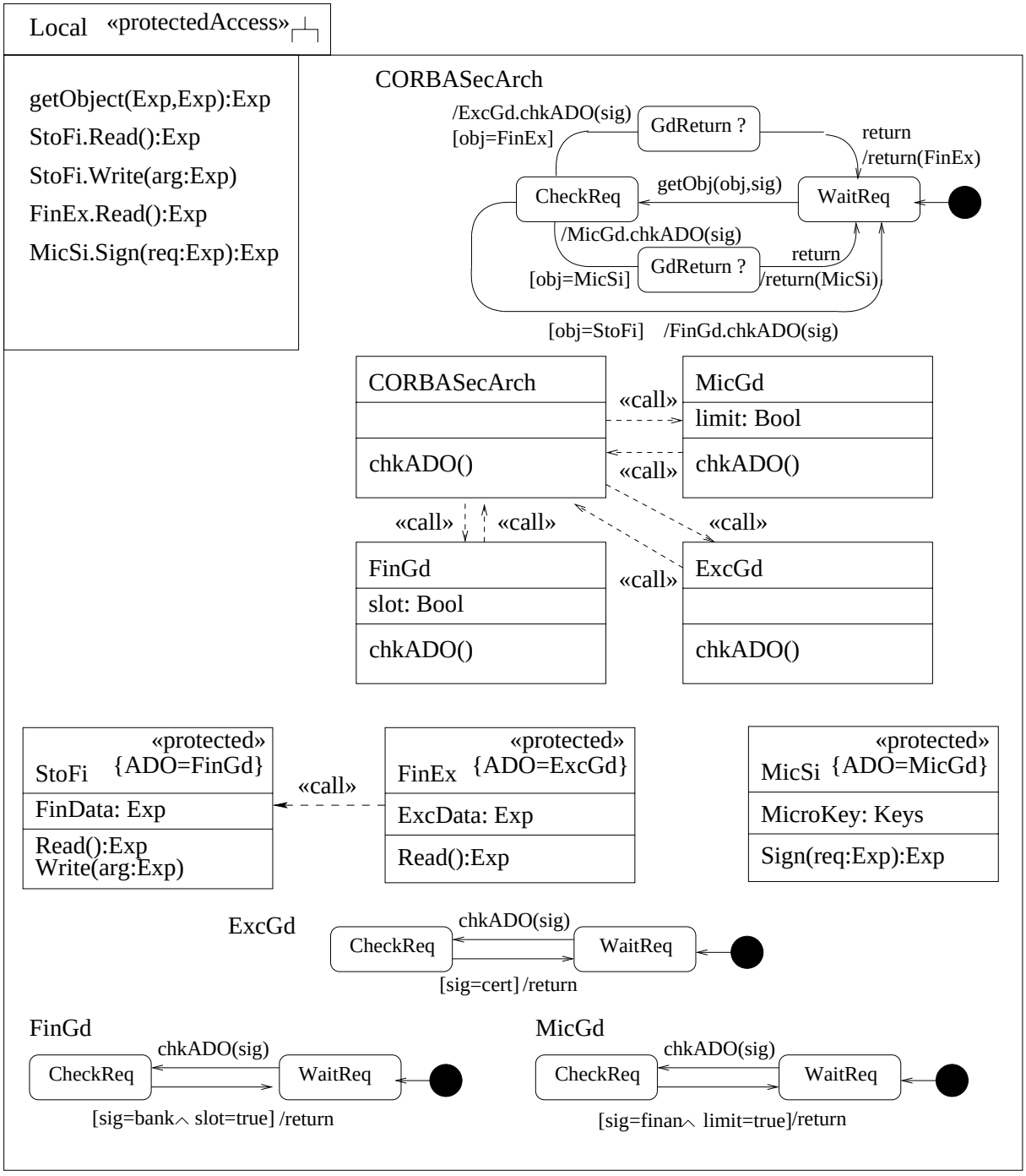

Figure 7: Subsystem