Einführung

Ziel: Ingenieurmäßige Entwicklung von Kommunikationssoftware

- **Unterscheidung zur norm. Softwareentwicklung durch:**
  - systematische Entwurfsmethodik (nicht ausgeprägt)
  - mehrere Spezifikationsebenen
  - Komplexität der Protokolle
  - Nebenläufigkeit, Nichtdeterminismen
  - hohe Zuverlässigkeitsanforderungen
  - Einfluss der Standardisierung
  - mehrfache Implementierung der Protokolle
- **Interne Ereignisse**
  - Innerhalb des Protokolles oder Dienstes auftretende Ereignisse die von außen nicht sichbar sind.
  - z.b. Fehler wie: Verbindungsabbruch (entspricht einer spontanen Transition in einem Zustansautomaten)
- **Nichtdeterminismus** tritt auf bei
  - Nebenläufig
  - interne Ereignisse
  - wenn keine Aussage über Reihenfolge der Ereignisse
  - bei Verhaltensalternativen nach Ereigniss
  - ermöglicht:
    - Vereinfachte Darstellung
    - Höhere Flexibilität

Auflösung von Nichtdeterminismen erst in der Implementierung

Dienstspezifikation

- **"Was"-Spezifikation**
  - Was soll für ein Dienst erbracht werden
  - Häufigvernachlässigt
  - Schnittstelle von außen betrachtet

Die Dienstspezifikation beschreibt die Eigenschaften des bereit- gestellten Dienstes und die Art und Weise seiner Nutzung.

- **Interessiert:**
  - Dienstnutzer
  - Verifizierer
  - Test-Ingenieur

- **Techniken der Dienstbeschreibung**
  - textuell
  - Zeitablaufdiagramme
  - Zustandsdiagramme
  - Parameter-Tabellen
- **Elemente der Dienstspezifikation**
  - Auflistung von (Teil-) Diensten und Dienstprimitiven
  - Abhängigkeiten zwischen Dienstprimitiven
  - lokales / globales Verhalten
  - interne Ereignisse
  - Nichtdeterminismen
  - Parameter und Abhängigkeiten zwischen ihnen

SAP - Service Access Point (Dienstzugangspunkt)

- **lokales Verhalten**
  - beschreibt zulässige Interaktionen an einem Dienstzugangspunkt
- **globales Verhalten**
  - beschreibt Verhalten zwischen verschiedenen SAPs

Protokollspezifikation

- **"Wie"-Spezifikation**
  - Wie soll der Dienst erbracht werden
  - Interne Umsetzung des Dienstes
  - "Wie"-Spezifikation ist nötig, weil Protokolle *mehrfach* implementiert werden *müssen*
    - verschiedene Betriebssysteme
    - unterschiedliche Ablaufumgebungen

Die Protokollspezifikation beschreibt die Art und Weise, wie die Instanzen in einem Protokoll miteinander kommunizieren, um den spezifizierten Dienst zu erbingen.

- **Elemente der Prorokollspezifikation**
  - Datenformatbeschreibung
    - i.d.R. mit ASN.1
  - Ablaufbeschreibungen
    - verhaltensorientiert
      - Zustandsautomaten
    - ablauforientiert
      - Zeitablaufdiagramme
      - MSC
  - Nichtdeterminismen
  - interne Ereignisse
  - lokale Aktionen

Beschreibunsmethoden und Beschreibungstechniken

Dienen der Beschreibung des **Protokollablaufs**

Beschreibungsmethoden vs. Beschreibungstechniken

- **Beschreibungsmethoden**
  - Methoden für die Beschreibung
  - Grundlage für die Modelle der Beschreibunstechniken
  - Bsp:
    - endliche Zustandsautomaten
    - Petri Netze
  - **KEINE vollständige Beschreibung**
- **Beschreibungstechniken**
  - Spezifikationssprachen für eine *formale* Beschreibung
  - Bsp:
    - SDL
    - Lotos
  - Nutz eine formale Semantik
  - **WEITGEHEND volständige Beschreibung**

Beschreibungsmethoden

- **Arten:**
  - Konstruktive Methoden
  - Deskriptive Methoden

Konstruktive Methoden

- Beschreibt Protokoll durch *abstraktes* Modell
- Interpretation durch semantisches Modell
- Quasi-Implementierung
- Bsp:
  - endliche Zustansautomaten
- **Vorteile**
  - direkte Unterstützung von:
    - Entwurf
    - Implementierung
  - Ausführliche Spezifikation
    - *Rapid Prototyping*
- **Nachteile**
  - keine Darstellung von Eigenschaften

Deskriptive Methoden

- beschreiben *Eigenschaften* eines Protokolls
- Unterlegen keine spezielle Interpretationsvorschrift
- Beschriebene Eigenschaften:
  - **Lebendigkeitseigenschaften**
  - **Sicherheitseigenschaften**
- Bsp:
  - Temporale Logiken
- **Vorteile**
  - direkte Formulierung gewünschter Eigenschaften
  - verifizierbar
- **Nachteile**
  - i. Allg. nicht entscheidbar, ob die Spezifikation das erlaubte Verhalten vollständig beschreibt
  - kein Prototyping
  - keine direkte Ableitung von Implementierungen

Klassifikation der Beschreibungsmethoden

Anmerkung: **FDT -> Field Device Tool**

konstruktiv und zustandorientiert

- Methoden:
  - endl. Zustandsautomaten
  - Petri-Netze
- FDTs:
  - SDL

konstruktiv und transitorientiert und prozessorientiert

- Methoden:
  - Prozessalgebren
  - Petri-Netze
- FDTs:
  - SDL
  - Lotos

konstruktiv und transitorientiert und ablauforientiert

- Methoden:
  - Zeitablaufdiagramme
  - Petri-Netze
- FDTs:
  - MSC

deskriptiv

- Methoden:
  - Temporale Logiken
- FDTs:
  - cTLA
- **Beschreibungsmethoden sind:**
  - Endliche Zustandsautomaten
  - Erweiterte endliche Zustandsautomaten
  - Petri-Netze
  - Algebraische Prozesskalküle
  - Temporale Logiken
  - Hybride Methoden

Endliche Zustandsautomaten

- Quintupel aus: (S,I,O,T,s0)
  - S - Zustände
  - I - Eingaben
  - O - Ausgaben
  - T - Zustandsüberführungsfunktion
  - s0 - Init-Zustand
- Transition t ist Zustandsübergang
- Wenn t != I, dann *spotane Transition*
- **Vorteile**
  - natürliche Beschreibung
  - einfache Darstellung
  - gut nutzbar für Implementierung
- **Nachteile**
  - Kommunikationsablauf kann nicht dargestellt werden
  - viele Zustände -> sehr groß und unübersichtlich

Erweiterte endliche Zustansautomaten

- Tupel aus: (S,C,I,O,T,s0,c0)
  - S - Zustände
  - C - Kontexte
  - I - Eingaben
  - O - Ausgaben
  - T - Zustandsüberführungsfunktion
  - s0 - Init-Zustand
  - c0 - Init-Kontext
- Transition t ist Zustandsübergang
- Wenn t != I, dann *spotane Transition*
- Kontext entspricht *wertbelegung der Variablen*
- **Vorteile**
  - natürliche Beschreibung
  - einfache Erstellung und Darstellung
  - weniger komplex als endl. Zustandsautomat
  - gut nutzbar für Implementierung
- **Nachteile**
  - Kommunikationsablauf kann nicht dargestellt werden
  - viele Zustände -> sehr groß und unübersichtlich
  - weniger geeigent für Dienstspezifikationen
  - Testfälle nicht direkt ableitbar

Beliebteste Beschreibunsmethode für Protokolle

Petri-Netze

Netz

- Tripel N=(P,T,F)
  - P - Plätze (Zustände,Bedingungen)
  - T - Transition (Ereignisse, Übergänge)
  - F - Flussrelation (Kanten zwischen den Plätzen/Zuständen)

Petri-Netz

- Quintupel N=(P,T,F,V,m0)
  - (P,T,F) - ein Netz

  - V - positive ganze Zahl die einer Kante zugeordnet ist
    - V - Kantenvielfalt (V=1 für gewöhnliche Netzte)
  - m0 - Anfangsmarkierung von P
- **Vorteile**
  - abstrakte Modellierung des Kontrollflusses
    - Unterstützung der Verifikation
  - Darstellung paralleler Abläufe
  - Nachvollziehen von Abläufen durch Verfolgen des Markenflusses
    - Prototyping
- **Nachteile**
  - sehr abstrakte Darstellung
    - Problem: Fehlerlokalisierung
  - sehr komplex bei vielen Zuständen
  - keine oder aufwendige Darstellung wichtiger Protokollelemente
    - Aufbau PDUs, Reihenfolgeüberwachung u. a.

Petri-Netze werden für die Protokollbeschreibung in der Praxis

- **Nutzung von Petri-Netzen**
  - Leistungsanlyse
  - Verfikation

Prozessalgebren

- **Prozessalgebren**
  - modellieren das Verhalten von Systemen durch Menge wechselwirkender Prozesse
  - nur das äußere Verhalten der Prozesse wird betrachtet.
  - Verallgemeinerung der klassischen Automatentheorie
  - Beschreibung der Systeme über die Kooperation kleinerer Komponenten, die **Prozesse**.
- **Grundelemente**
  - Aktion
  - Kompositionsoperatoren
- **Aktion**
  - ist: atomare, synchrone Interaktion mit einem *anderen* Prozess
  - werden durch *Label* repräsentiert
- **Label**
  - Ports über die interagiert wird
  - Keine Unterscheidung zwischen Ein-/Ausgabeereignissen
- **Vorteile**

  - exakte und vollständige Beschreibung
  - Unterstützung der formalen Verifikation
  - Unterstützung Testfallableitung
- **Nachteile**
  - sehr abstrakte Darstellung
  - große Distanz zur Implementierung
    - mehr Freiheitsgrade für Implementierung
  - hoher Grad an Formalität
    - Probleme in der Nutzerakzeptanz

Sehr begrenzte Nutzung für praktische Protokollentwicklung

Temporale Logiken

Temporale Logiken sind Erweiterungen der Aussagenlogik und der Prädikatenlogik durch Operatoren, die die Formulierung von Aussagen mit Bezug auf die Zeit gestatten.

- nicht die absolute Zeit interessiert, sondern die zeitliche Folge, in der sich die Dinge ereignen
- beschreiben Eigenschaften, die das Protokoll erfüllen soll
- Einhaltung der Eigenschaften im Protokollentwurf wird durch temporallogisches Schließen oder Model Checking überprüft

Wichtigste deskriptive Beschreibungsmethode für Kommunikationsprotokolle !!!

- **Unterscheidung der Eigenschaften**
  - Sicherheitseigenschaften
    - safety properties
  - Lebendigkeitseigenschaften
    - liveness properties
- **Sicherheitseigenschaften**
  - Bestimmt unerwünschte Ereignisse werden *nicht* eintreten
    - Beispiel: Keine Dateneinheit geht verloren
  - Erfüllt wenn nichts passiert
- **Lebendigkeitseigenschaften**
  - sicherstellen, dass Ereignisse auch wirklich eintreten
  - beschreiben was passieren *muss*
    - Beispiel: Nach Verbindungsaufbau wird Datentransfer-Phase erreicht und die Daten übertragen
- **Arten temporaler Logiken**
  - *Lineare temporale Logiken*

        * modellieren Verhalten in lineare Folge von Zuständen
        * nur zeitliches Verhalten
        * Erlaubt Aussagen zu Gegenwart und Zukunft
        * Nutzung: *Software Verfikation*
    - *Verzweigende temporale Logiken*
        * alternatives Verhalten in einem Zustand möglich
        * Verschiedene zeitliche Abläufe
        * Zustandsbaum
        * Nutzung: *Hardware Verfikation*

- **Vorteile**
    - explizite Spezifikation der zu erfüllenden Eigenschaften
    - Allgemeingültigkeit kann formal bewiesen werden
        * Inkonsistenzen können aufgedeckt werden

- **Nachteile**
    - Übergang zur Implementierung kompliziert
        * fehlende Werkzeugunterstützung
    - hoher Einarbeitungsaufwand
        * Vorkenntnisse erforderlich

Für praktischen Protokollspezifikationen kaum genutzt !!!

Zusammenfassung

- **FSM** reine FSM werden nur begrenzt genutzt
- **EFSM** Populärste Beschreibungsmethode für Protokolle
- **Petri-Netze** werden in der Praxis kaum genutzt
- **Prozessalgebren** sehr begrenzte Nutzung
- **Temporale Logiken** werden in der Praxis kaum genutzt
    - Anwendung in Kombination mit anderen Methoden
    - (Model Checking)

Beschreibungstechniken

- **Anforderungen**
    - präzise, eindeutig, vollständig
        * Vermeidung von Mehrdeutigkeit
    - Implementierungsunabhängig
        * Unterstützt verschiedene Ablaufumgebungen
    - verständlich
        * Vermeidung von Missinterpretation
    - modular, veränderbar
        * ermöglicht modifikation und erweiterung

- **Verschiedene Techniken**
  - SDL
    - **Wichtigste Technik**
    - graphische/sprachliche Notation
  - MSC
    - ablauforientiert
    - graphische Notation
    - meist im Kontext von *SDL* eingesetzt
  - Estelle
    - sprachliche Notation
    - *kaum noch Benutzt*
  - Lotos
    - Prozessalgebra
    - sprachliche Notation
  - ASN.1
    - bevorzugte Notation für Datenformate
  - UML2
    - *keine* formale Beschreibungstechnik!
      - wg. fehlender Semantik
    - *immer beliebter*
  - TTCN
    - Informale Testnotation

SDL - Specifikation and Description Language

Ziel: *Formale Beschreibung von Telekommunikationssystemen*

- Objekt-Orientiert
- SDL-2000: Aktuelle Version
  - Bietet *Agentenkonzept*
- **SDL-Notationen**
  - SDL/GR - graphische Notation
  - SDL/PR - textuelle Notation
  - *Besitzen gemeinsame Sprachelemente*
- **Agenten**
  - beschreiben aktive Komponenten
  - sind endl. Zustansautomaten
  - *besitzen:*
    - Identifikation
    - Lebensdauer
    - Warteschlange für *Stimuli*

- **Arten von Agenten**
  - Blöcke
    - enthält *Blöcke* und *Prozesse*
    - *nebenläufige Ausführung*
  - Prozesse
    - enthält *nur* Prozesse
    - *alternierende Ausführung* - Nur eine Transition
  - System
    - äußerster Block
- **Stimuli**
  - Ereigniss das Zustandsübergang auslöst
  - Erstes Ergeigniss in Warteschlange

Kommunikation zwischen Agenten über

- **asynchronern Nachrichtenaustausch mit Signalen**
  - Hat *Namen*
  - impliziete *Senderidentifikation*
  - Austausch der Signale über **Kanäle**
- **Kanäle**
  - Zuverlässige, reihenfolge bewahrende Übertragung
  - *Übertragungsarten*
    - verzögernd
    - verzögerungsfrei
- **Gates**
  - Endpunkte der Kanäle
  - externe Kommunikationspunkte der Agenten
  - implizite/explizite Gates
- **entfernte Prozeduraufrufe Client/Server-Prinzip**
- **gemeinsame Variable**
- **Komposite Zustände und Zustandsaggregation**
  - erst seit SDL 2000
  - Komposite Zustände
    - hierarchische Struktur von Zuständen
    - alle Zustände eine gemeinsame Warteschlange
    - Agent kann sich in mehreren Teilzuständen befinden
    - es wird immer nur eine Transition ausgeführt

- Zustandsaggregation
  - Partitionierung eines kompositen Zustands in mehrere (komposite) Zustände, die nach dem Interleaving-Prinzip ausgeführt werden
- **Ausnahmen**
  - erst seit SDL 2000
  - beschreiben unerwartetes Verhalten, z. B. Fehlersituationen
  - Verzweigung zu einer Ausnahmebehandlung
    - explizit beschrieben
- **Objekt-Orientierung**
  - *Typen* - Klassen
    - Agententypen
    - komposite Zustandstypen
    - Signaltypen
    - einfach Datentypen
  - *Instanzen* - entspricht Objekte
    - System
    - Block
    - Prozess
    - Signal
- **Formale SDL-Semantik**
  - *Statische Semantik*
    - nur für Kernsprache definiert
    - Ableitung eines abstrakten Syntaxbaums
  - *Dynamische Semantik*
    - Ableitung eines Verhaltensmodell aus Syntaxbaum
    - Interpretation als ASM-Kode
    - SDL-to-ASM-Compiler

MSC - Message Sequence Charts

- Dient der Visualisierung/Darstellung von Kommunikationabläufen in Systemen.
- ursprünglich *nicht* als FDT konzipiert
- Stellt *Interaktionen* zwischen Komponenten eines Systems sowie der Umgebung dar
- Integration in UML-2
- über Sequenzdiagramme
- Nicht an eine bestimmte Spezifikationssprache gebunden
- bevorzugt aber im Umfeld von *SDL* genutzt

- **Anwendungen**
  - Entwurf von Kommunikationabläufen
  - Dokumentation
  - Testfallbeschreibung
- **MSC unterstützt:**
  - formale Semantik (Prozessalgebren)
  - High-level MSC (HMSC)
  - Datentypen
  - entfernte Methodenaufrufe
  - Objekt-Orientierung
- **MSC-Notationen**
  - MSC/GR - graphische Notation
  - MSC/PR - textuelle Notation
- **Grundelemente**
  - Instanzen - Systemkomponenten
  - Nachrichten - Interaktion
- **Zeit im MSC**
  - entlang der Instanzachse schreitet die Zeit voran
    - es entsteht eine zeitliche Ordnung
  - Senden und Empfangen sind asynchrone Ereignisse
- **Darstellungsmöglickeiten in Bezug auf Nachrichten**
  - Überholen von Nachrichten
  - Verlust von Nachrichten
  - Finden von Nachrichten
- **Verfügbare Timerarten**
  - Start Timer
  - Stop/Reset Timer
  - Timeout
- **Bedingungen**
  - Beschreiben Systemzustände oder Vorbedingungen
- **Systemzustände**
  - shared all - globale Zustände für alle Instanzen
    - Können in verschiedenen MSCs enthalten sein!
  - shared - Zustände die nur *einige* Instanzen

  - lokale Zustände

- **Inline-Ausdrücke**
  - loop - Zyklen
  - opt - wahlweise Ausführung
  - exc - Ausnahmebehandlung
  - alt - Ausführungsalternativen
  - par - parallele Ausführung

- **High-level MSC**
  - Kombination von MSC zu komplexeren Beschreibungen
  - Referenzen auf MSC
  - Start/Stopp-Symbole

ASN.1 - Abstract Sytanx Notation One

- *Unabhängig von der FDT-Entwicklung* hat sich ASN.1 als Beschreibungssprache für Datenformate in der Telekommunikation durchgesetzt
- flexibler
- weniger aufwendig
- ISO/ITU Standard
- es sind zahlreiche Serialisierungsregeln definiert
- häufig eingesetzt im Protocol Engineering (X.500, LDAP, GSM, UMTS, etc.)
- es gibt grafische ASN.1 Editoren und APIs/Toolkits/Compiler
- **Ziele**
  - Beschreibung von Datenformaten
  - Genutzt bei DNS und Netzmanagment
- **Verschiedene Serialisierungsregeln**
  - Verschiedene Basic Encoding Rules (BER)
  - Canonical Encoding Rules (CER)
  - Distinguished Encoding Rules (DER)
  - Packed Encoding Rules (PER)
  - XML Encoding Rules (XER)
  - Generic String Encoding Rules (GSER)
- **Datentypen**
  - *atomic-types*
    * boolean
    * integer

        * enumerated
        * real
        * bit string
        * octet string
        * null
        * printable string
        * utf8-string
    - *structered-types* - (aus atomic-types zusammengesetzt)
        * SEQUENCE
        * SEQUENCE OF
        * SET
        * SEt OF
        * CHOICE

- **BER**
    - Basieren auf TVL-Kodierung
    - TVL: Type Length Value

Probleme formaler Beschreibungstechniken

- Protokolle werden meistens ad hoc entworfen,
- selten formal beschrieben,
- formale Beschreibungen werden meistens nur ergänzend genutzt.
- **Gründe für den begrenzten Einsatz formaler Beschreibungstechniken**
    - Nutzerakzeptanz
        * Nutzen formaler Techniken nicht ausreichend sichtbar
        * hoher Zeitdruck verhindert Einarbeitung
        * hoher Aufwand für Entwicklung, Validation von Spezifikationen
        * mangelnde Werkzeugunterstützung
        * unzureichende Effizienz generierter Implementierungen
    - Einarbeitungsaufwand
        * Einarbeitung in Sprache und semantisches Modell erforderlich
    - Entwicklungsaufwand
        * Entwicklungsaufwand formaler Beschreibungen auf der Grundlage informaler Beschreibungen ist beträchtlich
        * Umfang: 2000 – 10000 Zeilen
    - **Techniken mit graphischer Präsentation (SDL, MSC, UML)**
    - Verfügbarkeit anwendbarer Werkzeuge
        * noch keine ausreichende Werkzeugunterstützung
        * Prototypen vor allem im akademischen Umfeld
        * hoher Einarbeitungsaufwand
        * fehlende Unterstützung des gesamten Entwicklungsprozesses

  - Durchgängigkeit der Techniken
    * i.d.R. keine durchgängige Unterstützung des gesamten Protokoll- entwicklungsprozesses
    * viele Validationstools erfordern spezifische Eingabenotationen
  - zweimalige „Beschreibung“ der Protokolle
    * Spezifikation + Kodierung –> zu aufwendig
  - Fehlen einer Methodik
    * methodischer Rahmen für eine FDT-basierte Protokollentwicklung
    * analog OSI-Testmethodologie
    * Bewertungsmetriken
  - Verfügbarkeit formaler Beschreibungen
    * nur wenige formale Beschreibungen, insbesondere von Internet- Protokollen
    * Spezifikationen häufig erst nach den ersten Implementierungen verfügbar
    * häufig im akademischen Umfeld entstanden
    * wenn formale Spezifikationen verfügbar, dann nur als Komplement zur informalen Spezifikation

Entwicklung

Entwicklungschritte

- Kommunikationsprotokolle werden zum größten Teil in Software realisiert. - Kommunikationsoftware
- Protokollentwicklung entspricht im Kern der Softwareentwicklung
  - ABER: einige wichtige Unterschiede
- **Einteilung in mehrere Schritte - Wasserfallmodell**
  - Anforderungsanlyse
    * Anforderungsspezifiaktion
  - Dienst u. Protokollentwurf
    * Dienst u. Protokollspezifikation
  - Protokollverfikation
    * verfizierte Spezifikation
  - Leistungsvorhersage
    * optimierter Entwurf
  - Implementierung
    * Kode
  - Test
    * getestete Kommunikationsoftware
  - Installation/Integration
- **Anforderungsanlyse**
  - Spezifikation der Anforderungen an das Protokoll

  - meistens informal
  - häufig Berücksichtigung verschiedener (Firmen-) Interessen
- **Dienst u. Protokollentwurf**
  - 2 Schritte im Idealfall
    * Entwurf des Dienstes - nicht immer explizit ausgeführt
  - Entwurf des Protokolls
- **Protokollverfikation**
  - überprüft die funktionale Korrektheit des Entwurfs
    * Wird der spezifizierte Dienst wirklich erbracht ?
    * Widerspruchfreiheit des Entwurfs
  - *Programmverifikation/Konsistenzprüfungen in der Praxis kaum genutzt!!*
- **Leistungsvorhersage**
  - Prüft, ob der Entwurf angestrebten Leistungsvorgaben, z. B. bezüglich der Dienstgüte, gerecht werden kann
  - Finden von Schwachstellen im Implementierungsentwurf
- **Implementierung**
  - Abbildung der Dienst- und Protokollspezifikation in die Zielumgebung
  - Erarbeitung einer Implementierungsspezifikation
- **Test**
  - Konformitätstest
    * Übereinstimmung einer Implementierung mit der Spezifikation
    * Zertifizierung
  - Interoperabilitätstest
    * Zusammenarbeitsfähigkeit von Implementierungen
  - Leistungstest
    * Einhaltung von Leistungsvorgaben
  - Robustheitstest
    * Verhalten der Implementierung bei fehlerhaften Eingaben
- **Installation/Integration**
  - *sudo apt-get install myprotocol*
  - USE IT, BITCH!1ELEVEN

Besonderheiten der Protokollentwicklung

- mehrere Spezifikationsebenen
- spezifische Protokollverifikation
- Rolle der Standardisierung
  - Notwendigkeit Konformitätstest
- mehrfache Implementierung

  - Implementierungsspezifikation
  - Notwendigkeit eines Interoperabilitätstest
- **Spezifikationsebenen**
  - Dienstspezifikation
    * „Was“-Spezifikation
    * entspricht der Anforderungsspezifikation der Softwareentwicklung
  - Protokollspezifikation
    * „Wie“-Spezifikation
    * abstrakte Implementierung der Dienstspezifikation
    * erforderlich wegen mehrfache Protokollimplementierungen
    * Spezifikation von Protokolloptionen
  - Implementierungsspezifikation
    * bedingt durch Implementierungsunabhängigkeit der Protokollspezifikation
      · Anforderungen des Zielsystems / Ablaufumgebung
      · Auswahl von Protokolloptionen

Entwurf

- Adhoc-Protokollentwurf
  - in der Praxis vorwiegend praktiziert
    * weil kaum praktikablen systematischen Methoden
  - Grund: Viefalt der Anforderungen
- **Systematischer Protokollentwurf**
  - theoretische Ansätze
  - Vorteil: Absicherung von Protokolleigenschaften (z. B. Sicherheit, Lebendigkeit) durch den Entwurf
    * keine Verifikation notwendig !!!
- **Analyitsche Entwurfsmethoden**
  - Schrittweiser Protokollentwurf ausgehend von einer formulierten Anforderungsdefinition.
    * entspricht der traditionellen Vorgehensweise
  - Bsp: Entwurfsregeln
    * z. B. aus MSC
  - Bsp: Entwurfsmuster
    * z. B. SDL-Pattern
- **Erstellung der Spezifikation**
  - Aufwendig und anspruchsvoll
  - Problem: Erstspezifikationen sind in der Regel informal
    * formale Spezifikationen entstehen häufig erst wesentlich später
    * häufig zu spät

  - Problem: Subjektiver Einfluss des Spezifizierers
    * Grund: Komplexität
    * kaum autorisierte formale Spezifikationen !!!
  - Üblicherweise mit Prototyping - von abstrakt zu konkret

Verfikation oder Validation

Die Protokollverifikation hat die Aufgabe, die logische Korrektheit und die richtige Funktionsweise des dokumentierten Protokollentwurfs zu prüfen.

Begriffsbestimmung

Der Begriff der Verifikation wird in der Literatur unterschiedlich verwendet !!! alternative Verwendung Verifikation - Validation

- Protokollvalidation
  - Prozess der Bewertung der funktionalen und nichtfunktionalen Eigenschaften des Protokollentwurfs und seiner Implementierung bezüglich der Nutzeranforderungen
    * *Entwickeln wir das richtige System ?*
- Protokollverifikation
  - Formaler Nachweis der Korrektheit, Vollständigkeit und Konsistenz des Protokollentwurfs, repräsentiert durch die Spezifikation
    * *Entwickeln wir das System richtig ?*

Prüft auf und mit

- **Formaler Nachweis**
  - der spezifizierte Dienst wird erbracht
  - die Spezifikation korrekt, vollständig und konsistent

Voraussetzung für mehrfache Implementierungen des Protokolls !!!

- **Korrektheit**
  - Funktionelle Korrektheit des Entwurfs und der Erbringung des spezifizierten Dienstes innerhalb einer endlichen Zeit
- **Vollständigkeit**
  - Berücksichtigung aller möglichen Ereignisse und Optionen
- **Konsistenz**
  - innere Widerspruchsfreiheit des Entwurfs
    * z. B. Deadlock-Freiheit, Livelock-Freiheit

Prototyping

Unter Prototyping versteht das Ausführbarmachen der Spezifikation zu Validationszwecken.

- FDT-Compiler unterstützen Prototyping-Implementierungen
  - Abbildung z.B. in C++ und Verbinden mit Laufzeitsystem, das im Kern die FDT-Semantik umsetzt
- Prototype muss nicht notwendigerweise vollständig sein !!!
- Prototyping ist wiederholbar
  - entsprechend der Iterationsstufe der Spezifikationsentwicklung

Prototyping ist die am meisten genutzte Validationsmethode in der praktischen Protokollentwicklung !!!

Formen der Protokollverifikation

Verifikation der allgemeinen Eigenschaften

- Überprüfung von Eigenschaften, welche erfüllt sein müssen
  - vor allem Lebendigkeitseigenschaften
  - zielt vor allem auf die Konsistenz und Vollständigkeit des Entwurfs

Verifikation der speziellen Eigenschaften

- Eigenschaften, die durch die Semantik des Protokolls bestimmt sind
  - insbesondere Sicherheitseigenschaften
  - prüft ob der Protokollentwurf den spez. Dienst auch wirklich erbringt

Wichtige Protokolleigenschaften

Eigenschaft	Erklärung
keine nicht ausführbaren Aktionen	Protokoll enthält *keine* Aktionen die *nie zur Ausführung* kommen.
Deadlock-Freiheit	Protokoll gelangt *nie* in einen Zustand den *nicht mehr verlassen* kann.
Livelock-Freiheit	Protokoll gelangt *nie* in einen Zustand dem *unproduktive Zyklen* ausgeführt
Fehlertoleranz und Resynchronisation	Protokoll kehrt nach einem Fehler oder einer abnormalen Situation in einen
Vollständigkeit	Protokoll enthält keine nichtspezifizierten Ereignisse, d.h. alle Ereignisse sin
Terminierung	Protokoll erreicht immer den/die Endzustände; bei zyklischen Protokollen e

Verifikationsmethoden

- **Modellbasierte Verifikation**
  - Überprüft die Korrektheit, Vollständigkeit und Konsistenz der Spezifikation durch Beweistechniken, über semantischen Modell.
    - Erreichbarkeitsanalyse
    - statische und dynamische Analyse von Petri-Netzen
- **Deduktive Verifikation**
  - Basiert auf Nutzung von Axiomen und Interferenzregeln. Nutzen Syntax der Beschreibungstechnik als formale Basis, was syntaktisch Schlussfolgerungen über die Korrektheit der Spezifikation zulässt.
    - temporallogisches Schließen
- **Hybride Techniken**
  - Versucht Vorteile zu kombinieren.
    - Model Checking
- **Manuelle Techniken**
  - Beweistechniken, um z. B. Protokollinvarianten zu überprüfen
    - Einhaltung von Nachrichtenfolgen
    - Puffergrenzen
  - aufwendig, langwierig und selbst sehr fehleranfällig
  - **praktisch kaum genutzt !!!**

Erreicharkeitsanalyse

Unter Erreichbarkeitsanalyse wird die vollständige Untersuchung aller erreichbaren Zustände und Transitionen der durch die FSM beschriebeneno Protokollinstanz verstanden.

- am häufigsten genutzte Verifikationstechnik für FSM/EFSM- basierte Protokollbeschreibungen
- Erzeugung eines *Zustandsraums* der Instanz, der alle erreichbaren Zustände umfasst
- *Erreichbarkeitsgraph:* Graph, der ausgehend vom Initialzustand der Protokollinstanz alle erreichbaren Zustände und Zustandsübergänge enthält
- Die Erreichbarkeitsanalyse besteht aus zwei Schritten
  - Erzeugung des Erreichbarkeitsgraphen
  - Analyse seiner Eigenschaften
- Zwei Vorgehensweisen
  - vollständige Erzeugung des Graphen mit anschließender Analyse
  - On the fly –Techniken
    - verbinden Erzeugung des Graphen mit der Analyse von Eigenschaften
    - brechen die Analyse ab, wenn Fehler entdeckt werden
    - vermeiden das Abspeichern einer Vielzahl von Zuständen

Ziele

Ziel der Erreichbarkeitsanalyse ist es zu untersuchen, ob jeder Zustand des Protokolls erreicht wird und die Ausführungspfade dahin die geforderten Korrektheits-, Vollständigkeits- und Konsistenzkriterien erfüllen.

- Die Erreichbarkeitsanalyse unterstützt die:
  - Verifikation allgemeiner Eigenschaften
  - Verifikation spezieller Eigenschaften

Zustandsraumexplosion

Die analysierten Zustände müssen bei der Erreichbarkeitsanalyse abge- speichert werden.

- Unterstützung Backtracking
- Vergleich von Zuständen, um Mehrfachanalysen zu vermeiden
- Großer Speicherplatzbedarf !!!
  - besonders bei EFSM; bei Automaten mit mehren Teilautomaten (Interleaving)
- Zustandsraumexplosion
  - 10^9 Zustände selbst bei kleinen Protokollen !
- **Vollständige Erreichbarkeitsanalyse nur bei kleinen Protokollen möglich!!**

Bewältigung des Zustandsraumproblems

- Dekomposition und Partionierung
  - separate Analyse von Protokollphasen bzw. –teilen
    - Abhängigkeiten können nicht immer aufgelöst werden !!!
- Projektion
  - Modellvereinfachung
    - Zusammenfassung von mehreren Zuständen zu einem Zustand
- Zufällige Suche
  - willkürliche Auswahl des nächsten zu analysierenden Zustands
    - bei sehr großen Systemen zweckmäßig, wenn keine sinnvolle Menge von Zuständen mehr abgespeichert werden kann
  - **kann nur Fehler aufdecken, nicht Abwesenheit beweisen !!**

Implementierung

Realisierung des Protokolls in einer konkreten Ablaufumgebung entsprechend der Vorgaben der Protokollspezifikation

- aufwendig, kompliziert
- Vielzahl von Entscheidungen, die Effizienz und Korrektheit des Protokolls beeinflussen
- starke Abhängigkeit von der Implementierungsumgebung
- subjektiver Einfluss des Implementierers

Arten der Protokollimplementierung

- Manuelle Implementierung
  - hauptsächliche Vorgehensweise für reale Implentierungen
- Automatische Implementierung
  - FDT-basiert
  - meistens Prototyping
    - zum Zwecke der Validation
  - seltener reale Implementierungen

Implementierungsentwurf

- **Implementierungsentwurf**
  - Abbildung der Vorgaben der Dienst- und der Protokollspezifikation in die gegebene Ablaufumgebung
    - erforderlich weil
      - logisches Modell der Protokollspezifikation nicht erhalten bleiben muss
      - Protokollspezifikationen einige Entscheidungen der Implementierung überlassen
- **Berücksichtigung der Randbedingungen**
  - Protokollimplementierung ist immer auf ein konkretes Zielsystem ausgerichtet
    - Betriebssystem, Implementierungssprachen, vorhandene Bibliotheken
- **Lokale Implementierungsentscheidungen**
  - Entscheidungen, die aus Gründen der Implementierungsunabhängigkeit erst beim Implementierungsentwurf getroffen werden
  - **Können Konformität und Interoperabilität stark beeinflussen !!!**
- **Wahl des Prozessmodells**
  - wichtige Entscheidung des Implementierungsentwurfs
  - legt fest, wie die Implementierung in die Prozess-Struktur der Ablauf- bzw. Betriebssystemumgebung abgebildet wird
    - muss nicht zwingenderweise eine Eins-zu-Eins-Abbildung sein

- **Arten von Prozessmodellen**
  - Server-Modell
    - geradlinige Umsetzung
    - Eine Instanz
    - langsamer
    - einfach zu machen
  - Activity Thread-Modell
    - mehrere prallele Instanzen
    - schneller
    - schwerer umszusetzen
- **Implementierungsspezifikation**
  - dokumentiert Implementierungsentwurf
  - Grundlage für die Kodierung des Protokolls
    - Arbeitsdokument des Implementierers
  - grundsätzlich auf ein bestimmtes Zielsystem ausgerichtet
  - **häufig nicht explizit ausgeführt !!!**
  - **explizit erforderlich für automatische Implementierung !!!**

Kodierung

- **Kodierung des Protokolls**
  - Programmkode des Protokolls lässt sich i. Allg. relativ geradlinig aus der Spezifikation ableiten
  - weitere Funktionen erforderlich
    - Prozessverwaltung
    - Pufferververwaltung
    - Zeitverwaltung
- **Ablaufumgebung**
  - Ablaufumgebung hat einen hohen Anteil an der Protokollausführung
  - ca. 60 %
  - **Ein Protokoll kann nicht unabhängig von der Ablaufumgebung implementiert werden !!!**
- **Möglichkeiten der Einbindung eines Protokolls in die Ablaufumgebung**
  - Integration in das Betriebssystem
    - üblicherweise transportorientierte Instanzen
  - Realisierung als Anwendungsprozesse
    - anwendungsorientierte Instanzen
  - Nutzung protokollspezifischer Ablaufumgebungen
    - Portierbarkeit
    - einheitliche Schnittstellen
    - Berücksichtigung protokollspezifischer Erfordernisse

Test

Ein Test ist ein Experiment, in dem untersucht wird, ob ein Objekt bestimmte, erwartete Anforderungen erfüllt !

Der Protokolltest hat die Aufgabe zu prüfen, ob das implementierte Protokoll die Vorgaben der Spezifikation umsetzt.

- komplementär zur Protokollverifikation
  - bezieht sich dabei nur auf eine vorliegende Implementierung
- Es gelten grundsätzlich ähnliche Aussagen wie beim Softwaretest:
  - Ziel des Testens ist es, Fehler in der Implementierung aufzudecken !!!
  - Ein Test kann nur das Vorhandensein von Fehlern nicht aber ihre Abwesenheit nachweisen !!!
- **Wichtigste praktische Validationsmethode für Protokolle !!!**

Arten des Protokolltests

- Entwicklungsbegleitende Tests
  - Debugging
- Konformitätstest
  - Übereinstimmung mit Spezifikation
- Interoperabilitätstest
  - Zusammenarbeitsfähigkeit von Implementierungen
- Leistungstest
  - Leistungsverhalten der Implementierung
- Robustheitstest
  - Verhalten der Implementierungen bei falschen Eingaben

Konformitätstest

Ziel des Konformitätstests ist es, zu überprüfen, ob eine gegebene Implementierung eines Protokolls der zugrundeliegenden Spezifikation des Protokolls entspricht. **In der Praxis vor allem ein Test auf Einhaltung von Protokoll-Standards**

Interoperabilitätstests

Der Konformitätstest allein kann die Zusammenarbeitsfähigkeit verschiedener Protokollimplementierungen, ihre Interoperabilität, nicht gewähr- leisten. * **Gründe** * Umfassende Konformitätstests sind in der Praxis aus Kostengründen kaum möglich !!! * lokale Implementierungsentscheidungen * Wahl unterschiedlicher Protokolloptionen

Testarten

- White Box (oder Glass Box Test)
  - Der Quellkode ist dem Testingenieur zugänglich
- Black Box Test
  - Der Quellkode ist dem Testingenieur nicht zugänglich
    - nur Test bzgl. des nach außen sichtbaren/beobachtbaren Verhaltens !!!
- Grey Box Test
  - Dem Testingenieur sind beschränkt Informationen über den Quellkode zugänglich *z. B. Strukturinformationen