FORSOFT

Bayerischer Forschungsverbund
Software Engineering

ADL-Workshop München 2001

Technische Universität München
31. Juli 2001

Manfred Broy
Michael Gnatz
Ingolf Krüger
Frank Marschall
Gerhard Popp
Andreas Rausch
Wolfgang Schwerin
(Hrsg.)

Gesponsert durch:
Bayerische Forschungsstiftung
FORSOFT - Bayerischer Forschungsverbund Software Engineering
Projekt ZEN - Zentrum für Technik, Methodik und Management der Software- und Systementwicklung

Agenda:

Zeit:	Thema:	Vortragender:
9:00 - 9:20	Begrüßung	Prof. Dr. Manfred Broy
9:20 - 9:30	Kurze Vorstellung des Fallbeispiels	Frank Marschall
9:30 - 9:50	Modellierung mit der B-Method	Prof. Dr. Johannes Siedersleben (sd&m Research)
9:50 - 10:10	Modellierung mit Rapide	Gerhard Popp
10:10 - 10:30	Modellierung mit Wright/ACME	Maurice Schoenmakers
10:30 - 10:50	*Kaffepause*	*Alle*
10:50 - 11:10	Modellierung mit Focus	Frank Marschall
11:10 - 11:30	Modellierung mit UML RT	Wolfgang Prenninger
11:30 - 11:50	Modellierung nach Hofmeister et. al	Tobias Miller (4Soft GmbH)
11:50 - 12:10	Modellierung mit UML/Together	Andreas Günzler
12:10 - 12:30	Modellierung mit Catalysis	Alexander Ziegler
12:30 - 14:00	*Mittagessen*	*Alle*
14:00 -15:00	Gemeinsame Diskussion: Wunschliste für die eigene ADL • Brainstorming, Sammeln von Stärken und Schwächen der verwendeten Beschreibungstechniken • Ordnen der Wünsche	*Alle,* Moderation: Dr. Ingolf Krüger und Frank Marschall
15:00 - 16:00	Gemeinsame Diskussion: Konzepte von ADLs: • Identifikation von Konzepten • Ordnen von Konzepten • Gemeinsame Zuordnung ADLs/Konzepte/Anwendungsfälle	*Alle,* Moderation: Dr. Ingolf Krüger und Frank Marschall

ADL-Workshop

Ziele

Ziel des Workshops ist es, die Konzepte bestehender Architekturbeschreibungssprachen („Architectural Description Languages“ bzw. ADLs) zu identifizieren und zu vergleichen. Von besonderem Interesse ist es hierbei, einerseits die Mächtigkeit einzelner ADLs zu untersuchen und andererseits ihre Eignung für den praktischen Einsatz zu prüfen.

Im folgenden werden eine Reihe von Fragestellungen zu ADLs vorgestellt, welche eine Klassifikation ggf. erleichtern können. Diese sollten jedoch lediglich als Anregungen zur Auseinandersetzung mit ADLs verstanden werden.

Um die Ausdrucksmächtigkeit, und somit das potentielle Anwendungsfeld einer ADL besser abschätzen zu können, bietet sich eine Untersuchung folgender Fragestellungen an:

- Was ist das zugrundeliegende (Meta-)Modell der ADL? Wie werden Komponenten in der ADL beschrieben und welche Arten von Relationen existieren zwischen Komponenten (Uses-Beziehung, Datenfluss, Kontrollfluss, Kommunikationskanal)?
- Welche Konzepte zur Typisierung und Parametrisierung von Daten, Komponenten, Kommunikationskanälen und/oder Nachrichten existieren?
- Wie wird das Verhalten von Architekturelementen beschrieben? Welche Konzepte zur Kommunikation zwischen Architekturelementen werden unterstützt (synchrone, asynchrone Kommunikation, gemeinsame Speicherbereiche). Wie werden Echtzeitanforderungen, Nebenläufigkeit, etc. modelliert?
- Existieren Konzepte zur konsistenten Verfeinerung und Abstraktion von Architekturbeschreibungen?
- Existiert eine mathematische Fundierung, die ggf. eine Verifikation der Korrektheit der Architekturbeschreibung bzw. bestimmter Eigenschaften erlaubt?

Darüber hinaus lohnt es sich, sich mit Fragen zur Praxistauglichkeit einzelner ADLs auseinander zu setzen. Denkbare Fragestellungen hierzu sind:

- Wird/wurde die ADL bereits in der Praxis eingesetzt und setzt sie bereits existierende Notationen ein.
- Existiert eine graphische Notation?
- Existieren verschiedene Architektursichten zur Beschreibung von Eigenschaften, die für die Realisierung der Architektur besonders relevant sind (z.B. Package-Strukturen, Topologie, Systemumgebung)?
- Inwieweit wird eine mögliche physische Verteilung einzelner Architekturkomponenten berücksichtigt?
- Wie gut lassen sich nicht-funktionale Eigenschaften (z.B. Antwortzeiten oder Durchsatz) durch die ADL beschreiben?
- Existieren methodische Ansätze und Vorgehensweisen zur Verwendung der ADL?
- Inwiefern unterstützt die ADL einen Entwurf auf hoher Ebene?

Neben der Klärung dieser Fragestellungen sollen die Teilnehmer Erfahrungen mit dem Umgang mit ADLs und der Modellierung „echter“ Systeme mit Hilfe formaler Beschreibungssprachen sammeln.

Vorgehen

Jeder Teilnehmer bzw. jede Teilnehmergruppe wählt sich eine spezifische ADL aus (vgl. Liste am Ende der Workshop-Beschreibung). Die Teilnehmer machen sich in einem Zeitraum von ca. einer Woche mit dieser ADL vertraut. Anschließend versucht jeder Teilnehmer das im folgenden Abschnitt beschriebene System mit Hilfe dieser Beschreibungssprache zu modellieren. Auf die Modellierung der GUI des Systems kann hierbei verzichtet werden.

Im Anschluss stellen die Teilnehmer im Rahmen eines gemeinsamen Workshops jeweils „ihre“ ADL und die fertige Modellierung des Systems vor. In der gemeinsamen Diskussion sollen die im vorigen Abschnitt angeführten Fragen zur Mächtigkeit und Verwendbarkeit von ADLs für die einzelnen Beschreibungssprachen erörtert werden.

Das zu modellierende System

Modelliert werden soll das von sd&m-Research zur Verfügung gestellte Telecom-Billing-System (TBS) zur Erstellung von Telefonrechnungen. Hierzu kann auf die vorhandene Dokumentation zugegriffen werden. Als „letzte Instanz“ bei Unklarheiten in der Spezifikation des Systems dient der Quellcode. Hierdurch soll sichergestellt werden, dass alle Beteiligten das selbe System modellieren und keinen Design-Contest veranstalten. Das TBS wurde um eine zusätzliche Komponente Billing erweitert, die mit Fremdsystemen der Kunden kommuniziert. Abbildung 1 zeigt den groben Aufbau des Systems, wobei die Komponenten für die kein Quellcode existiert gestrichelt eingezeichnet sind.

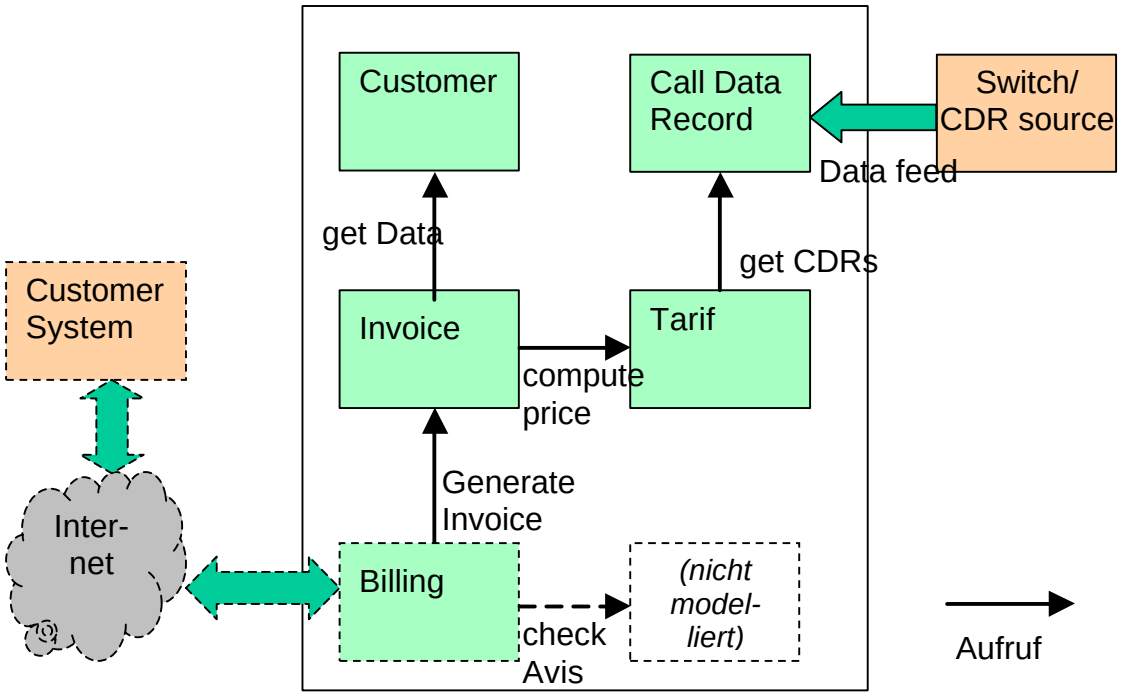


Abbildung 1: Wesentliche Komponenten des TBS

Das bestehende TBS-Beispiel von sd&m-Research

Folgende Annahmen liegen dem Modell zu Grunde:

- Jedem Kunden ist ein Tarif zugeordnet.
- Ein Anruf wird immer genau über ein Tarifmodell abgewickelt.

- Die Rechnungserstellung erfolgt über sämtliche Anrufe eines Kunden in einem bestimmten Zeitraum.

Momentan ist ein einziges Tarifmodell vorgesehen (Typ A); weitere können dem System nach Laune hinzugefügt werden. Das Tarifmodell Typ A ist an die im Mobilfunk üblichen Tarife angelehnt. Es hat folgende Elemente, die jeweils für einen bestimmten Wochentag und eine bestimmte Zeitspanne wirksam sind (z.B. Montag bis Freitag von 8:00h bis 18:00h):

- Preis pro Minute
- Dauer des ersten Takts (z.B. 60 Sekunden: für jeden Anruf bis 60 Sekunden wird der volle Takt abgerechnet)
- Dauer der Folgetakte (z.B. 1 Sekunde: Sekundengenaue Abrechnung)

Entscheidende Vereinfachung im Vergleich zur Realität: Es gibt genau einen Tarif, der nicht zwischen Orts- und Ferngesprächen sowie Fest- und Mobilnetz unterscheidet (eine entsprechende Erweiterung des Systems wäre aber ohne grundsätzliche Schwierigkeiten möglich).

Für jeden Anruf erzeugt die Vermittlungsanlage (Switch) einen Datensatz (Call Data Record/CDR), für den mit Hilfe des gültigen Tarifs ein Preis berechnet wird. Diese CDRs fallen sofort nach dem Anruf an und werden zunächst in eine Datenbank geschrieben (der Switch als Quelle der CDRs ist in der Aufgabe als gegeben zu betrachten). Der CDR enthält Information über die wesentlichen Merkmale des Anrufs:

- Telefonnummer des Anrufers (Origin)
- Telefonnummer des Angerufenen (Destination)
- Startzeitpunkt (Datum, Uhrzeit)
- Endzeitpunkt (Datum, Uhrzeit)

Bei der Erstellung der Rechnung wird folgendermaßen vorgegangen:

- Zunächst werden die CDRs für den entsprechenden Kunden und den Rechnungszeitraum aus der Datenbank geholt.
- Für jeden Anruf wird dann auf Basis des entsprechenden Tarifs der Preis berechnet.
- Die Rechnungssumme ergibt sich aus der Aufsummierung der Einzelposten.

Die wesentlichen Komponenten eines solchen Systems sind:

- Verwaltung der Anruf-Datensätze (CDRs),
- Tarifverwaltung und Preisberechnung für die CDRs,
- Kundenverwaltung,
- Rechnungserstellung.

Die Klassen per werden mit dem Quasar Data Interface (QDI), einem sd&m-eigenen Persistenzframework, persistent gemacht. Mit Hilfe des Quasar User Interface Frameworks (QUI) wird eine GUI zur Verfügung stellt. Zunächst sollte jedoch das System ohne Datenbank und GUI modelliert werden. Anschließend kann es dann entsprechend erweitert werden.

Erweiterung von TBS um eine Komponente zum Rechnungsversand

Um auch Konzepte wie Nebenläufigkeit und Kommunikation zwischen Systemen zu berücksichtigen, wird das TBS-Beispielsystem zusätzlich um eine (stark vereinfachte) Komponente zur Versendung der erstellten Rechnungen erweitert. Im Beispiel wird davon ausgegangen, dass die Rechnungen an (Geschäfts-)Kunden elektronisch an deren Finanzsysteme versandt werden können. Die Kunden erhalten für die Nutzung des eigenen Internetzugangs des Telecom-Anbieters sog. „Phone-Miles“, mit denen sie Rabat auf ihre Telefonrechnung bekommen können.

Die Abbildungen 2 und 3 stellen exemplarisch den möglichen Ablauf einer erfolgreichen bzw. nicht erfolgreichen Zahlung dar.

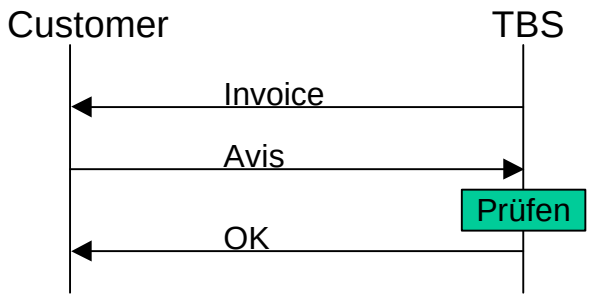


Abbildung 2: Erfolgreicher Zahlungsvorgang

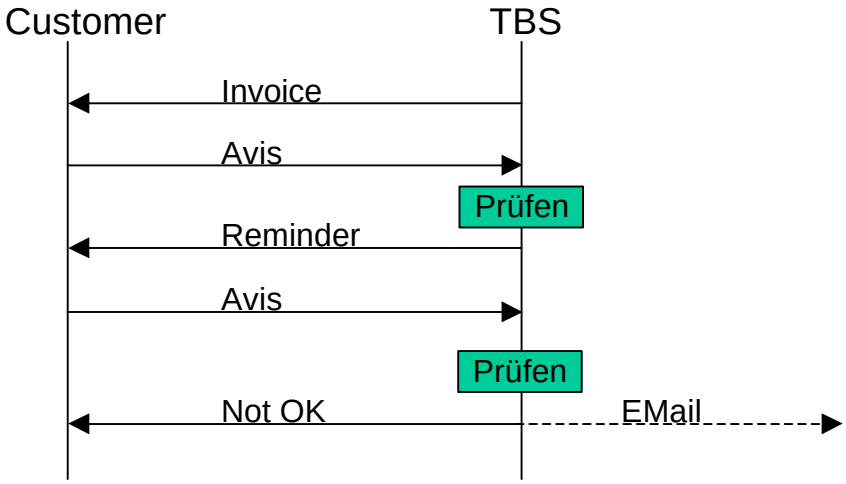


Abbildung 3: Erfolgloser Zahlungsvorgang

Die Komponente für den Versand von Rechungen lässt jeden Monat automatisch eine neue Rechnung erstellen und versendet die darin enthaltenen Daten mittels HTTP in einem SOAP-Aufruf „Invoice“ an den jeweiligen Kunden. Als Antwort sendet das Finanzsystem des Kunden eine Zahlungsankündigung (Avis), in dem es den Betrag, die Kontonummer und das Datum der Überweisung ankündigt. Der zu überweisende Betrag kann unter dem Rechnungsbetrag liegen, wenn der Kunde zusätzlich „Phone-Miles“ einlöst.

Das TBS prüft die Daten durch einen Aufruf an eine weitere Prüf-Komponente die ggf. auch Anwender involviert. Im Erfolgsfall sendet das TBS eine SOAP-Nachricht „Payment Accepted“ an das Kundensystem und zieht die Phone-Miles vom Konto des Kunden ab. Diese Komponente soll jedoch nicht mehr Teil der Beispielapplikation sein, für das hier zu modellierende System liefert sie lediglich nichtdeterministisch entweder den Wert „true“ oder „false“. Schlägt die Überprüfung des Avis fehl (z.B. durch eine ungültige Nachricht vom Kundensystem oder weil der Kunde nicht mehr genügend Phone-Miles hat), so wird eine Nachricht „Reminder“ an das Kundensystem, das alle Daten der Rechnung beinhaltet, geschickt. Das System des Kunden muss daraufhin erneut einen Zahlungsavis an das TBS senden.

Schlägt die Überprüfung dieses Avis ebenfalls fehl, so sendet das TBS eine Nachricht „Payment Not Accepted“ an das Kundensystem, um diesem zu signalisieren, dass die automatische Überweisung nicht erfolgen soll. Gleichzeitig wird das Problem per Email an einen Sachbearbeiter eskaliert, der nun die weitere Bearbeitung des Zahlungsvorgangs manuell abschließt. Das TBS verhält sich genauso, wenn es nach der Versendung einer Erinnerung nicht innerhalb von 48 Stunden eine gültige Antwort des Kundensystems erhält. Antwortet das Kundensystem innerhalb von 48 Stunden nicht auf eine „Invoice“-Nachricht, so wird eine Erinnerung verschickt; ein Kunde bekommt jedoch nie mehr als eine Erinnerung.

Die von der Billing-Komponente empfangenen und gesendeten SOAP-Nachrichten sind im einzelnen:

Gesendete Nachrichten:

Name	**Parameter:**

`Invoice`	`int invoiceID`
	`int amount`
	`date startDate`
	`date endDate`

`Reminder`	`int invoiceID`
	`int amount`
	`date startDate`
	`date endDate`
	`string comment`

`Payment Accepted`	
`Payment Not Accepted`	`string comment`

Empfangene Nachrichten:

Name	**Parameter:**

`Avis`	`int invoiceID`
	`int amount`
	`int amountOfPhoneMiles`
	`string bankAccountInfo`
	`date dateOfTransfer`
	`string comment`

Im Gegensatz zum Rest der Anwendung existiert für die Billing-Komponente kein Code. Die Signaturen der einzelnen SOAP-Nachrichten werden hier nicht explizit festgehalten, da sie für die Modellierung der Architektur eher unerheblich sind und als Black-Box-Komponenten angesehen werden können. Auch die von der Billing-Komponente verwendeten Technologien (SOAP-fähiger http-Server, XML-Parser, Datenbank etc.) werden hier nicht explizit beschrieben. Sie können als Black-Box-Komponenten angesehen werden, welche Methodenaufrufe in SOAP-Nachrichten umwandeln, bzw. beim Eingang einer SOAP-Nachricht entsprechende Methoden in der Billing-Komponente aufrufen.

Liste möglicher ADLs, ADL-Tools, etc.

Die folgende Liste stellt lediglich eine Auswahl möglicher Beschreibungssprachen und -techniken vor. Selbstverständlich ist jeder Teilnehmer frei bei der Wahl der eigenen Beschreibungstechnik, auch weniger formale Ansätze sind denkbar.

ACME	The ACME Homepage, http://www.cs.cmu.edu/~acme/
Aesop-Toolkit	R. Melton: The Aesop System: A Tutorial, http://www.cs.cmu.edu/afs/cs/project/able/www/aesop/html/tutorial/aesop-demo.html
ARMANI	R. Monroe: Armani Language Reference Manual, CMU Technical Report in preparation
Darwin	Jeff Magee, Naranker Dulay, Susan Eisenbach and Jeff Kramer, Specifying Distributed Software Architectures, Proc. of 5th European Software Engineering Conference (ESEC '95), Sitges, September 1995, LNCS 989, (Springer-Verlag), 1995, 137-153
Focus	M. Broy: “Specification and Development of Interactive Systems – FOCUS on Streams, Interfaces and Refinement“, Springer-Verlag, April 2000
UML (1)	C. Hofmeister, R. Nord, D. Soni: “Applied Software Architecture”, Addison-Wesley, 2000
UML (2)	Andreas Rausch: Dissertationsschrift „Evolutionärer Softwareentwurf“, 2001
Rapide	Rapied Homepage, Stanford University, http://pavg.stanford.edu/rapide/rapide-pubs.html
Spectrum	M. Broy et. al., The Requirements and Design Specification Language SPECTRUM – An Informal Introduction, Technischer Bericht TUM I9311-12, TU München 1993
UniCon	Gregory Zelesnik: The UniCon Language Reference Manual, Carnegie Mellon University, http://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/UniCon/reference-manual/Reference_Manual_1.html
Wright	Robert J. Allen: “A Formal Approach to Software Architecture”, Ph.D. Thesis, Carnegie Mellon University, CMU Technical Report CMU-C*S-97-144,* May 1997
Z	J. Michael Spivey. *The Z Notation: A Reference Manual*, second edition, Prentice-Hall, Englewood Cliffs, N.J., 1992

Sonstiges

Integrating Wright with other ADLs:

David Garlan, Zhenyu Wang: A Case Study in Software Architecture Interchange, *March 1998*

ADLs vergleichen/klassifizieren:

Nenad Medvidovic and Richard N. Taylor: A Classification and Comparison Framework for Software Architecture Description Languages, University of California, Irvine, 1997

Übersicht über verschiedene ADLs:

Architecture Description Languages, Carnegie Mellon SWE Institute, http://www.sei.cmu.edu/architecture/adl.html

M. Broy, E. Denert, C. Hoffmann, K. Renzel, M. Schmidt: Software Architectures and Design Patterns in Business Applications, Technical Report TUM-I9746

E. Di Nitto, D. Rosenblum: Exploiting ADLs to Specify Architectural Styles Induced by Middleware Infrastructures, ICSE 1999

B: Ein kurzer Überblick

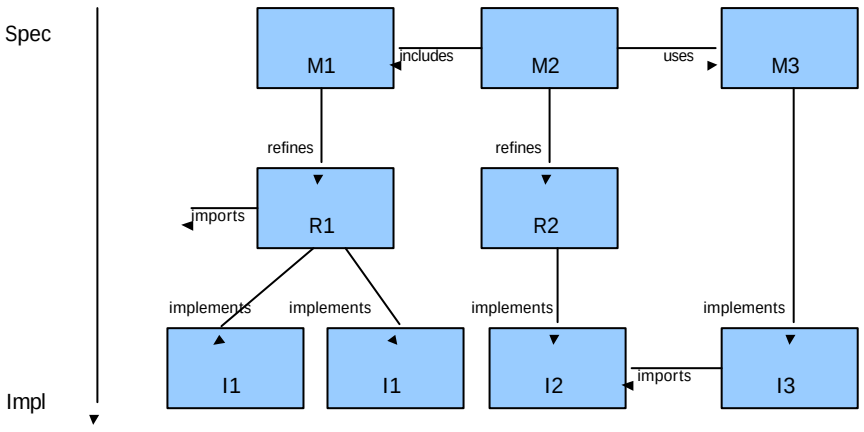
Johannes Siedersleben, sd&m Research, Juli 2001

Quelle The B-Book, Jean-Raymond Abrial, 1996
Assigning programs to meanings

Grundideen

- Mathematik: Mengen, Funktionen, Relationen, Folgen, Bäume
- Abstrakte Maschinen: Abstrakte Datentypen mit präziser Definition
- Schrittweise Verfeinerung von der Spezifikation zur Implementierung

B: Der Entwicklungsprozess



Abstrakte Maschinen in der Spezifikation

Machine My_Machine(list of parm)

USES Other_Machine

INCLUDES Still_Another_Machine

SETS enumerated oder deferred

CONSTANTS, PROPERTIES

VARIABLES, CONSTRAINTS, INITIALIZATION

OPERATIONS
PRE
nicht-ausführbarer Pseudo-Code:
Substitution
Generalized Substitution
Bounded Choice Substitution
Guarded Substitution

B: Standardfragen

- Metamodell: abstrakte Maschinen
- Synchrones, asynchrones Verhalten: abstracted away (?)
- Verfeinerung/Abstraktion: Refinement als Kern der Idee
- Mathematische Fundierung/Verifikation ist gegeben

- Einsatz in der Praxis: Pariser Metro; Beispiele im Buch:
  Invoice, Telephone Exchange, Lift Control, Database, Boiler Control
- Graphische Notation: nein (B-Book enthält viele Tabellen aber kein einziges Bild).
- verschieden Architektursichten: nein
- physische Verteilung: nein
- nicht-funktionale Eigenschaften: nein
- methodischer Ansatz: Specification, Refinement, Implementation
- Entwurf auf hoher Ebene: abstrakte Maschinen

Modellierung des Telekom-Billing-Systems mit Rapide

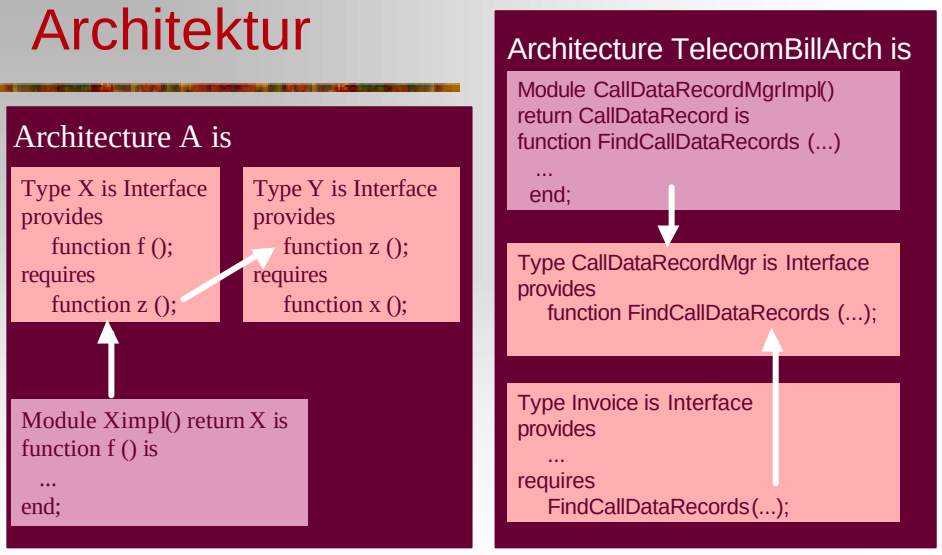
ADL-Workshop am 31.07.2001

Gerhard Popp

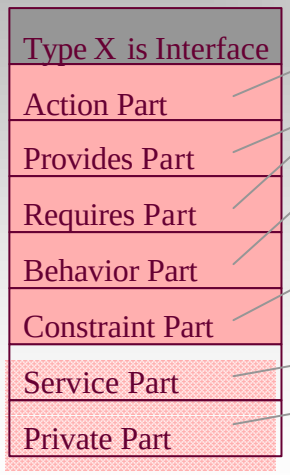
Übersicht über Rapide

- Entstanden an der Universität Stanford
- Entwicklungen am Projekt Rapide von 1990 bis 1997/98
- Kernsprachen
  - Type Language: Beschreibung von Komponentenschnittstellen
  - Pattern Language: Beschreibung von Ereignismustern
- Weitere Teilsprachen
  - Constraint Language: Bedingungen über Ereignisfolgen
  - Executable Language: Komponententypen und Architekturen als ausführbare Module
  - Architecture Language: Verknüpfung von Komponenten in strukturierter Weise
- Geeignet für das Prototyping großer, verteilter Systeme

Schnittstellen, Module und die Architektur



Type Language: Interfaces



- Ein- und ausgehende Aktionen als asynchrone Nachrichten
- Namen und Signaturen von Funktionen von und für andere Module
- Zustands- und Transitionsregeln, die auf dem aktuellen Zustand arbeiten und neue Nachrichten erzeugen
- Einschränkungen in dem Verhalten von Modulen, die die Schnittstelle implementieren. Definiert auf Parameter und Rückgabewerten
- Einteilung in Sub-Interfaces
- Sichtbarkeit nur innerhalb von Modulen, die die selbe Schnittstelle implementieren

Schnittstelle CallDataRecordMgr

```
Type CallDataRecordMgr is interface
action
    in   Add (record: CallDataRecord);
    out  Ack (record : CallDataRecord);
provides
    function FindCallDataRecords  (orig : PhoneNumber, f : DateTime, u : DateTime) return Container;
    function FindCallDataRecords  (orig : PhoneNumber, f : DateTime, u : DateTime, b: Boolean) return Container;
    ...
requires
    function GetStartTime () return DateTime;
    function GetEndTime () return DateTime;
    function GetOrigin () return PhoneNumber;
    ...
    -- Container-Funktionen, DateTime-Funktionen, PhoneNumber-Funktionen
    ...
behavior
    (?X in CallDataRecord)  Add(?X) => Ack(?X);
constraint
    match ( (?X  in CallDataRecord) Add(?X) -> Ack(?X) ) ^(*~);

end CallDataRecordMgr;
```

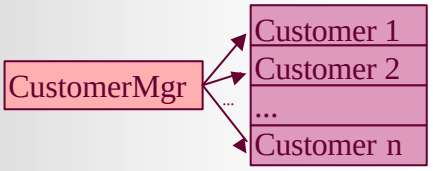

Executable Language: Modul CallDataRecordImpl

```
Module Ximpl () return X is
    function f () is
        ...
        z ()
        ...
    end;
begin
    ...
end;
```

- Ein Modul implementiert genau ein Interface
- Ein Interface kann von mehreren Modulen implementiert werden

```
Module CallDataRecordImpl() return CallDataRecord is
    ?start, ?end :  DateTime;
    ?orig , ?dest :  PhoneNumber; ...
    function GetStartTime () return DateTime is ... end;
    function GetEndTime () return ...
    function GetOrigin () return ...
    function GetDestination  () return ...
    ...
    function SetData (DateTime, DateTime,
                PhoneNumber, PhoneNumber) is ...
    function SetNull () is ...
begin
    -- Behavior-Regel aus Interface-Datei:
    -- Start => SetNull() => SetUnbilled() => SetData(...) ...
    SetNull();
    SetUnbilled();
    ...
    SetData(...);
    ...
end CallDataRecord;
```

Execute Language: Architektur




```
Architecture A () is InterfaceA
    x : X is Ximpl ();
            -- benötigt f ()
    y : Y is Yimpl ();
            -- liefert f ()
connect
    x.f () to y.f ();
constraint
    match (...);
end A;
```

```
CusM: CustomerMgr;
Cus : array [1..N] of Customer;
connect
    for i : Integer in 1..N generate
        CusM.GetName() to Cus[i].GetName();
    end generate;
```

???

```
CusM: CustomerMgr;
connect
    ?cus : Customer;
    Establish (?cus) => link (?cus);
    CusM.GetName() to ?cus.GetName();
    Abolish (?cus) => unlink (?cus);
```

- beschränkte Form eines Moduls
- kann eine Schnittstelle implementieren

Ausschnitt aus der TBS-Architektur

```
architecture TelecomBillingArch is BillingSystem
    CdrM : CallDataRecordMgr is CallDataRecordMgrImpl();
    CusM : CustomerMgr is CustomeMgrImpl;
    TarM : TarifMgr is TarifMgrImpl;
    InvG : InvoiceGen is InvoiceGenImpl;
connect
    ?Cdr : CallDataRecord;
    ?Cus : Customer;
    ?Inv : Invoice;
    ?orig, ?dest, ?num : PhoneNumber;
    ?from, ?until, ?issue : DateTime;
    ?elem : CallDataRecord;
    ?name, ?adr1, ?adr2, ?tarifid : String;
    ?billed : Boolean;
    ?tarif : Tarif;
    -- Verbindungen von InvoiceGen zu Invoice
    EstablishIIG(?Inv) => link(?Inv)
    InvG.GetCustomer() to ?Inv. GetCustomer();
    InvG.GetId() to ?Inv.GetId();
    -- Verbindungen von Invoice zu TarifMgr
    ?Inv.FindTarif(?tarifid) to TarM.FindTarif(?tarifid);
    -- Verbindungen von Invoice zu CallDataRecordMgr
    ?Inv.FindCallDataRecords(?orig, ?from, ?until) to
            CdrM.FindCallDataRecords( ?orig, ?from, ?until);
    -- Verbingungen von Invoice zu einzelnen CDRs
    EstablishCI(?Cdr) => link(?Cdr);
    ?Inv. GetStartTime() to ?Cdr.GetStartTime();
    ?Inv. GetEndTime() to ?Cdr.GetEndTime();
    ...
    -- Verbindungen von Invzu Tarif
    EstabilshTI(?tarif) => link(?tarif)
    ?Inv.ComputePrice(?Cdr) to ?tarif.ComputePrice(?Cdr);
    AbolishTI(?tarif) => unlink(?tarif);
    AbolishCI(?Cdr) => unlink(?Cdr);
    -- Verbindungen von Invoice zu CustomerMgr
    ?Inv.FindCustomer(?name) to
            CusM.FindCustomer(?name);
    ?Inv.FindCustomer(?num) to CusM.FindCustomer(?num);
    -- Verbindungen von Invoice zu einzelnen Customern
    EstablishCuI(?Cus) => link(?Cus);
    ...
    AbolishCuI(?Cus) => unlink(?Cus);
    AbolishIIG(?Inv) => unlink(?Inv)
end TelecomBillingArch;
```

Zusammenfassung

- Interfaces mit Requires-, Behavior- und Constraint-Regeln
- Tool-Unterstützung
- Grafische Beschreibungstechniken nicht vorgesehen
- Module sehr nah an der Implementierung
- Architektur bei großen Systemen aufteilen
- Keine explizite Objekterzeugung zur Laufzeit
- Rapide wird seit 1998 nicht mehr weiterentwickelt

ACME & Wright

Maurice Schoenmakers, schoenma@in.tum.de

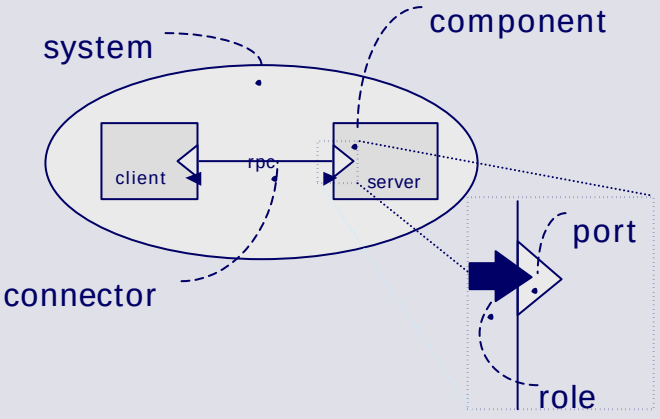
Acme:

An Architecture Description Interchange Language,
David Garlan, Robert T. Monroe, David Wile,
Proceedings of CASCON '97, November 1997.

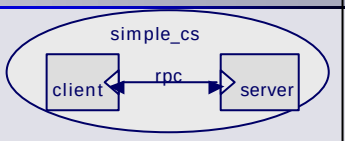
- **Generic** *Structure* **Centric ADL to support ADL Interchange**
- **Extensible Language Base for new ADL's**
- **Generic Platform for ADL independent tool development**

ACME Metamodel, Graphical

Graphical Representation, Structural Language



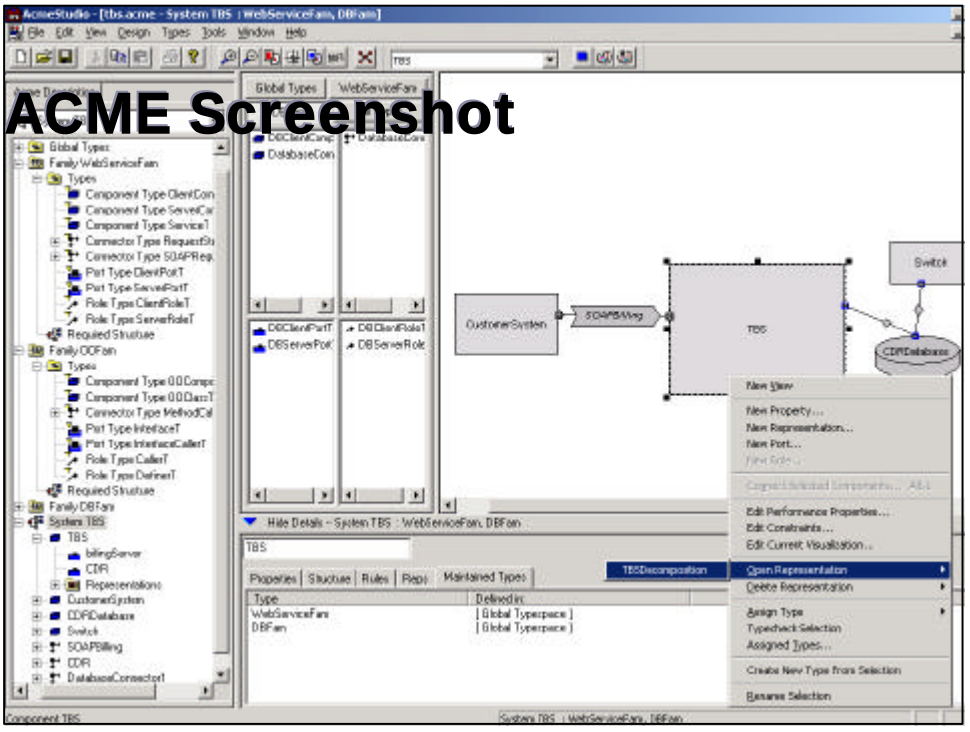
ACME Textual



Core Structural Syntax: Types & Properties

```
System simple_cs = {
     Component client    = { Port send-request    }
     Component server    = { Port receive-request }
     Connector rpc       = { Roles { caller, callee}
                             Properties {
                                max-roles : integer = 2;
                                protocol : WRIGHT = „...“}
                           }
     Attachments   { client.send-request    to  rpc.caller;
                     server.receive-request to rpc.callee;
                   }
}
```

ACME Screenshot



WRIGHT

WRIGHT:
Robert J. Allen. *Ph.D. Thesis,*
Carnegie Mellon University,CMU Technical Report CMU-CS-97-144, May 1997

Describes Behavior using CSP Processes

Components
- **Ports**
- **Computation**

Connectors
- **Roles**
- **Glue**

Static Architectures:
- **Styles (Behavior + Constraints)**
- **Configurations (Renaming + Attachments)**

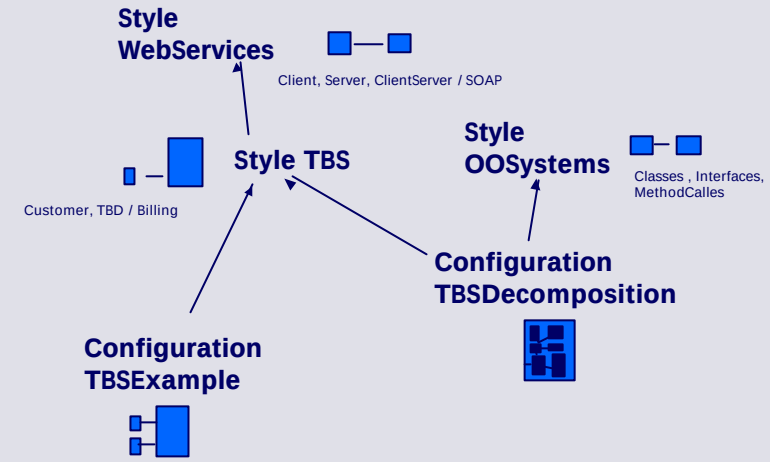
WRIGHT

Style **Client-Server**
Component **Client**
Port **p = request ® reply ® p** éù **§**
Computation **= internalCompute ® p.request ® p.reply ®** Computation éù **§**
Component **Server**
Port **p = request ® reply ® p [] §**
Computation **= p.request ® internalCompute ® p.reply ®** Computation **[] §**
Connector **Link**
Role **c = request ® reply ® c** éù **§**
Role **s = request ® reply ® s [] §**
Glue **= c.request ® s.request ®** Glue
[] s.reply ® c.reply ® Glue
[] §
Constraints
" s Î Component, " c Î Component : TypeServer(s) Ù TypeClient(c) Þ connected(c,s)
EndStyle

Configuration **Simple**
Style **Client-Server**
Instances **client : Client ; rpc : Link ; server : Server**
Attachments **client.p** as **rpc .c ; server.p** as **rpc. s**
EndConfiguration

WRIGHT TBS Model

Approach



WRIGHT WebServices & OO

Style **WebServices**

Interface Type **ClientT** = request ® reply ® ClientT éù §
Interface Type **ServerT** = request ® reply ® ServerT[] §

Component **ServerComponentT** (numClients: 1..)
Port **client**$_{1..numServers}$ = ServerT
Computation = (|| i : 1.. numClients ·
client$_i$.request ® internalCompute$_i$
® client$_i$.reply ® Computation) [] §

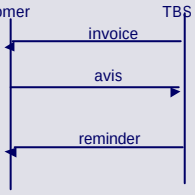
Component **ClientComponentT** (numServers: 1..)
Port **server**$_{1..numServers}$ = ClientT
Computation = (éù i : 1.. numServers · internalCompute$_i$ ® server$_i$.request ® server$_i$.reply ® Computation) éù §
Connector **RequestStreamConnectorT** ...
Connector **Type SOAPRequestConnectorT** : RequestStreamT ...
Constraints " c : Connectors · Type(c) = RequestStreamConnectorT...
EndStyle

Style **OOSystem**

Interface Type **CallerT**$_E$ = (éù e : E · e.request ® e.result ® CallerT$_E$) éù §
Interface Type **DefinerT**$_E$ = ([] e : E · e.request ® e.result ® DefinerT$_E$) [] §
Connector **MethodCall**(S : å) / / interface signature
Role **caller** = CallerT$_S$
Role **interface** = DefinerT$_S$
Glue = ([] e : S · caller.e.request ® interface.e.request
® interface.e.result ® caller.e.result ® Glue) [] §

Component **Class** (DEFINED: å *, USED: å *) / / signature sequences ....
...
EndStyle

WRIGHT TBS Billing Protocol



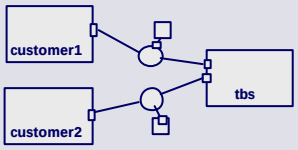
Interface Type **SOAPCustomerBilling = invoice ® firstAnswer [] § where**
firstAnswer = avis ® firstTry éù reminder ® secondAnswer
secondAnswer = avis ® secondTry éù NotOK ® SOAPCustomerBilling
firstTry = OK [] reminder ® secondAnswer
secondTry = OK [] NotOK ® SOAPCustomerBilling

Interface Type **SOAPTBSBilling = invoice ® firstAnswer éù § where**
firstAnswer = firstAvis ® firstTry [] 48hover® reminder ® secondAnswer
secondAnswer = secondAvis ® secondTry [] 48hover® NotOK ® SOAPCustomerBilling
firstTry = OK éù reminder ® secondAnswer
secondTry = OK éù NotOK ® SOAPCustomerBilling

Connector **SOAPBilling : SOAPRequestConnectorT**
Role **customer = SOAPCustomerBilling**
Role **tbs = SOAPTBDBilling**
Role **clock = wait48h ® alarm ® clock éù stop ® clock éù §**
Glue = **customer.avis ® clock.stop ® tbs.firstAvis ® afterFirstAvis [] $Glue_1$**
where **afterFirstAvis = [] customer.avis ® clock.stop ® tbs.secondAvis ® $Glue_2$ [] $Glue_2$**
$Glue_G$ = tbs.invoice ® clock.wait48h ® customer.invoice ® $Glue_G$
[] tbs.reminder ® clock.wait48h ® customer.reminder ® $Glue_G$
[] clock.alarm ® tbs.48hover ® $Glue_G$
[] tbs.OK ® customer.OK ® G [] tbs.NotOK ® customer.NotOK ® $Glue_G$

WRIGHT TBSExample

Configuration **TBSExample : TBSSystem**
Instances
customer1: CustomerSystemComponent
customer2: CustomerSystemComponent
tbs: TBSComponent(2)
billingA : SOAPBilling
billingB: SOAPBilling
clock1: Clock
clock2: Clock



Attachements
customer1.billing as **billingA.customer**
customer2.billing as **billingB.customer**
$tbs.billing_1$ as **billingA.tbs**
$tbs.billing_2$ as **billingb.tbs**
billingA.clock as **clock1.alarm**
billingA.clock as **clock2.alarm**

End **TBSSystem**

Positive

ACME:

- Detailed Structural Decomposition
- Tool Support
- Multiple Representations

WRIGHT : adds Behavior to ACME

- Protocols
  - Detailed description of technical details (message *transfer*)
- Modularization and Reuse:
  - Ports/Roles separation from Computation/Glue
  - Interface Compatibility
- Semantic Foundation
  - Easy to understand the CSP basics
- Easy rules to describe static architectures
  - (Easy?) mathematical formulation of constraints

Negative

ACME

- Unclear and semantically unsupported extension

WRIGHT

- No dynamic structures
  - Parameterization not enough,
  - No practical support for connection/component creation and deletion
- Unclear how to perform (protocol) refinements
  - semantically probably defined by CSP but no support by rules...
  - Unsatisfying data/state and Message contents description
- Separation of Connectors and Components?
  - connectors are here components but can not be combined?
- Support for Views
  - computation/glue contains always complete behavior
  - Missing relation between technical and functional views
  - Extension is not enough, to strict
- Hard to understand/describe the essential parts of the behavior
- Not able to describe concisely collaborations among multiple components
- Cannot describe non functional time based requirements

Conclusions

Structural decomposition is quite easy to use

But behavioral descriptions and constraints are

- **Semantically defined**
- **but still to hard to describe (math, no graphics, no collaborations),**
- **to use (to verbose in unimportant and to concise in important areas)**
- **and to check (rules and tools are missing)**

ACME/WRIGH Application:

Besides structural formulation the behavioral description is Useful especially to describe for technical protocols with static structures for people with mathematical knowledge

What's missing:

- **Pre existing libraries of styles, with predefined connectors**
- **Dynamic structures closer to real systems needed**
- **Standardized graphical representations**
- **Better support for refinement of protocols, ports etc.**
- **Easy to use refinement rules**

TBS-Modellierung mit FOCUS

Forsoft/ZEN ADL-Workshop
31.07.01
Frank Marschall

FOCUS - Kurzbeschreibung

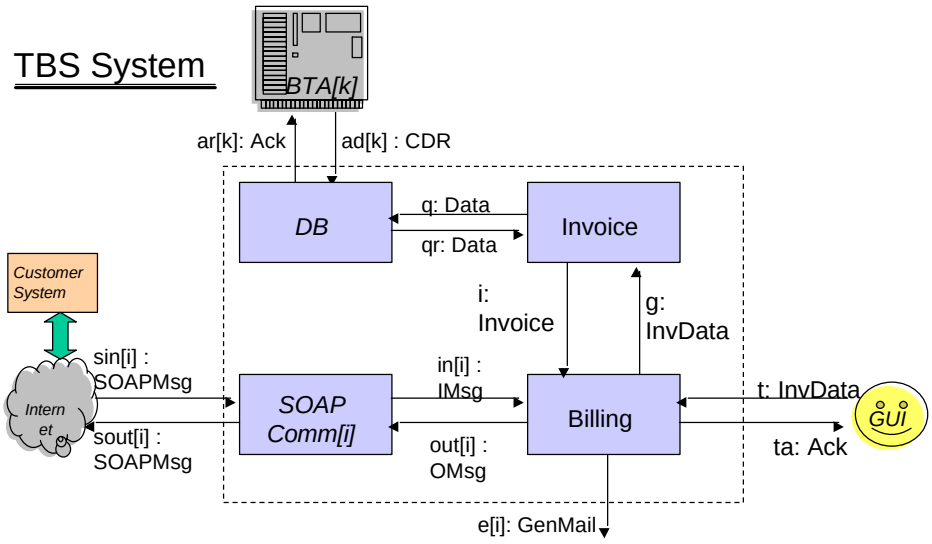
FORSOFT
ZEN

- Buch: Prof. M. Broy: „Specification and Development of Interactive Systems: FOCUS on Streams, Interfaces, and Refinement“, 4.4.00
- „Formal Description and Development Technique“
  - Gedacht als mathematisches Fundament für Beschreibungstechniken und formale Verfikation, nicht unbedingt als eigenständige Beschreibungstechnik
  - Aber: auch graph. Notationen und Tabellen
- Ströme
  - Ströme aus getypten Nachrichten fließen asynchron über Kanäle passenden Typs zwischen Komponenten
- Kanäle
  - Unidirektionale Verbindungen zwischen Komponenten
- Komponenten
  - Verarbeiten Ströme von Nachrichten (stromverarbeitende Funktionen)
  - Verschiedenen Specifications-Stile (A/G, Equational, Tables, Diagrams) zur Spezifikation des Verhaltens einer Komponente
  - Composite Specifications mit Black Box und Glass Box Views (ebenfalls graphisch oder textuell)

Modellierung

FORSOFT
ZEN

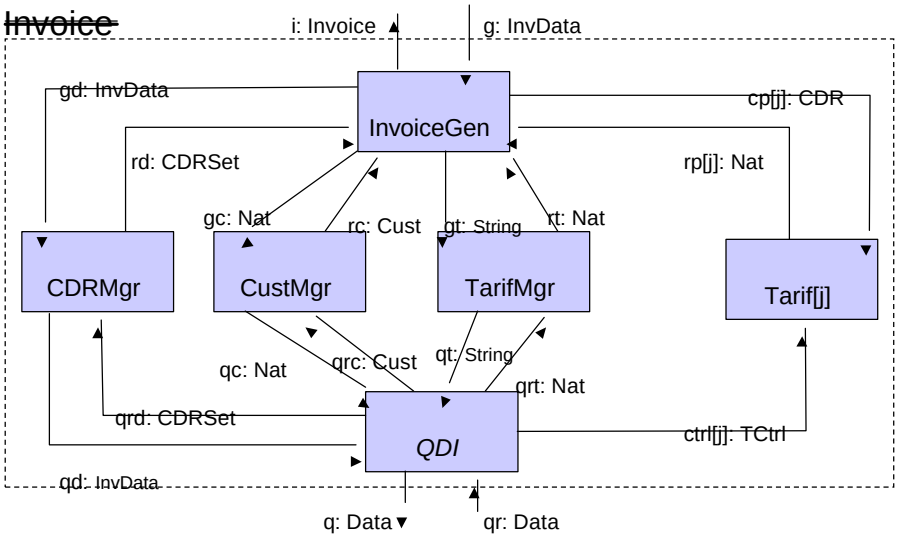
TBS System



Modellierung

FORSOFT
ZEN

~~Invoice~~

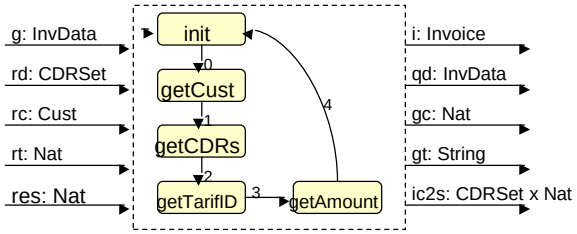


Modellierung

FORSOFT
ZEN

InvoiceGenCtrl timed

lcl tId ∈ Nat // Tarif ID
cdrs ∈ CDRSet
st ∈ Date // start time
et ∈ Date // end time
invId ∈ Nat // Inv. Id
univ p ∈ Nat // Phone No.
sd,ed ∈ Date // Start/Ende
c1, c2, ... ∈ CDR
name ∈ String // Cust. Name
. . .



	i	g	gd	rd	gc	rc	gt	rt	ic2s	res	tId'	cdrs'	st'	et'	invId'
0	√	(p, sd, ed)	√	√	p	√	√	√	√	√	∅	{}	sd	ed	invId
1	√	√	(p, st, et)	√	√	(p, name, custId, addr., baddr, tarifId)	√	√	√	√	tarifId	{}	st	et	invId
2	√	√	√	{c1, c2,...} ∈ CDRSet	√	√	tId	√	√	√	tId	{c1, c2,...}	st	et	invId
3	√	√	√	√	√	√	√	t	(cdrs,t)	√	tId	cdrs	st	et	invId +1
4	invId, m, st, et	√	√	√	√	√	√	√	√	m	tId	cdrs	st	et	invId

Eigenschaften

FORSOFT
ZEN

- Elemente: Stromverarbeitende Komponenten (Funktionen), getypte Nachrichten, gerichtete Kanäle
- Typisierung/Parametrisierung:
  - Nachrichten & Kanäle sind haben immer einen Typ
  - Komponenten: Replications und Parametrisierung
- Kommunikation:
  - generell nur asynchrone Nachrichten, synchron kann „simuliert“ werden
- Verhaltensbeschreibung
  - Elementare Spezifikationen: A/G, Equational, Tables, Diagrams
  - Composite Specifications: Verhalten ergibt sich aus elementaren Specs
- Konsistente Verfeinerung/Abstraktion
  - Behavioural Refinement
  - Interface Refinement
  - Conditional Refinement
- Einsatz in der Praxis: AutoFocus
- Graphische Notation: Ja, Black Box Views & Glass Box Views
- Methodischer Ansatz: Methodik-Bausteine vorhanden, Verfeinerung

Erfahrungen

FORSOFT
ZEN

- Problematisch für die Modellierung des Beispiels
  - Komponenten („Objekte“) zur Laufzeit instanzieren
  - Aufteilung von Objekten in Daten und Komponenten
  - Korrekte Formulierung bei Kommunikation mit vielen Partnern (Sheafs, d.h. Arrays von Kanälen) nicht immer einfach
  - Lediglich logische Architektursicht kann abgebildet werden
- Positiv
  - Unterspezifikation von Komponenten durch Angabe von Constraints
  - Verfeinerung mit graphischer Notation sehr übersichtlich
  - Sehr einfache Modellierung von Verhalten durch Automaten

FORSOFT
ZEN

ADL-Workshop

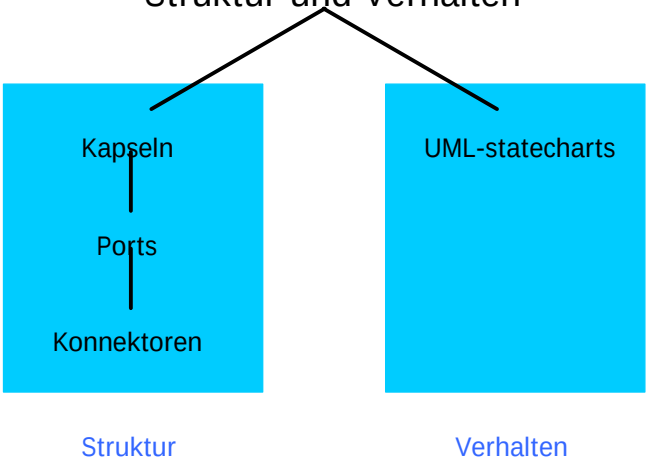
Modellierung des
Telecom-Billing-Systems
mit UML-RT.

31.07.2001

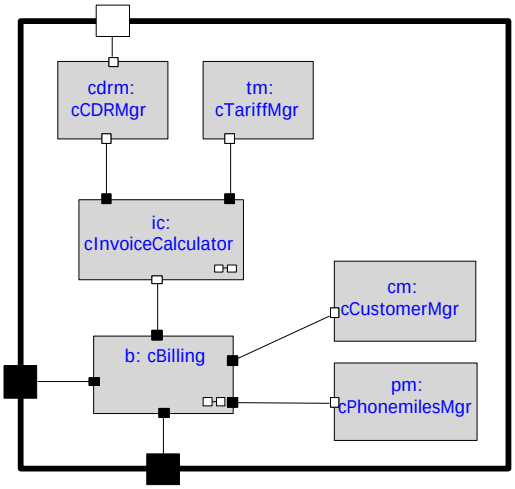
Wolfgang Prenninger

Das Komponentenmodell in UML-RT

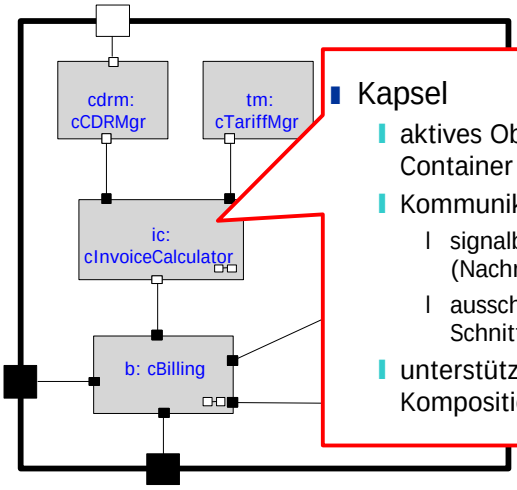
Beschreibungstechniken der UML-RT für Struktur und Verhalten



Struktursicht auf das Telecom-Billing-Systems

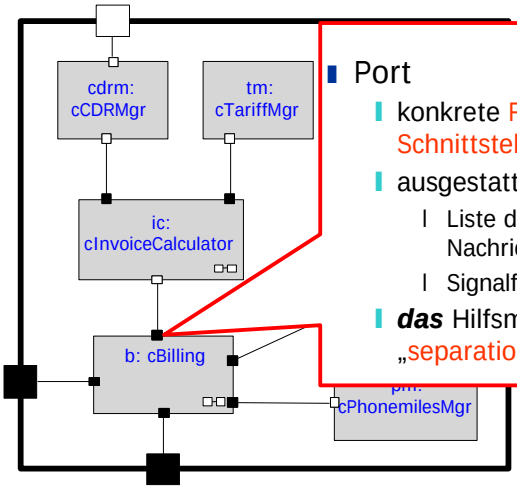


Struktursicht auf das Telecom-Billing-Systems



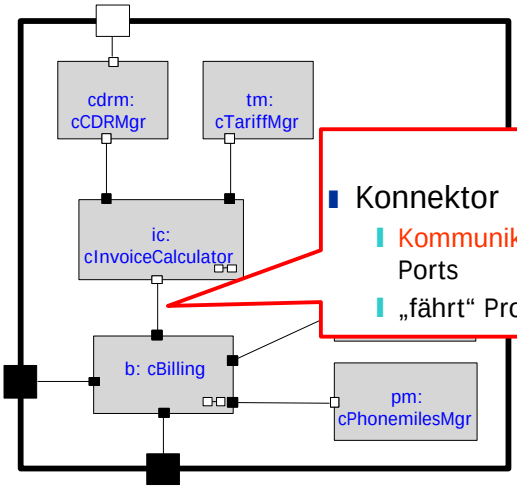
- Kapsel
  - aktives Objekt, oder „passiver“ Container
  - Kommunikation mit der Umgebung
    - signalbasiert (Nachrichtenaustausch, asynchron)
    - ausschließlich über Schnittstellenobjekte (Ports)
  - unterstützt hierarchische Komposition

Struktursicht auf das Telecom-Billing-Systems



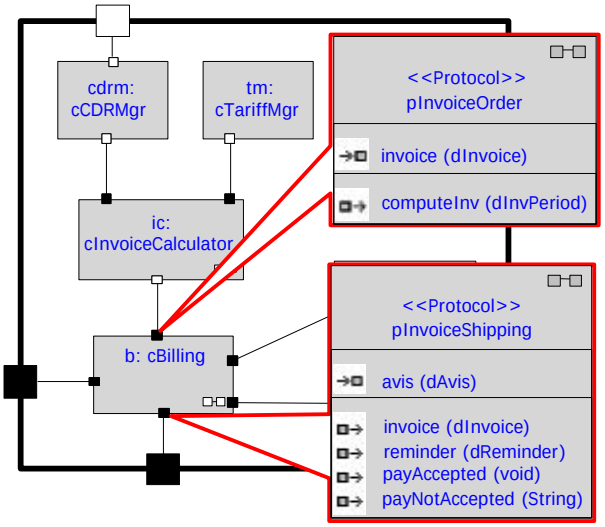
- Port
  - konkrete Realisierung einer Schnittstelle durch ein Objekt
  - ausgestattet mit einem Protokoll
    - Liste der ein-/ausgehenden Nachrichten/Signale
    - Signalfluß
  - ***das*** Hilfsmittel für „Kapselung“ und „separation of concerns“

Struktursicht auf das Telecom-Billing-Systems

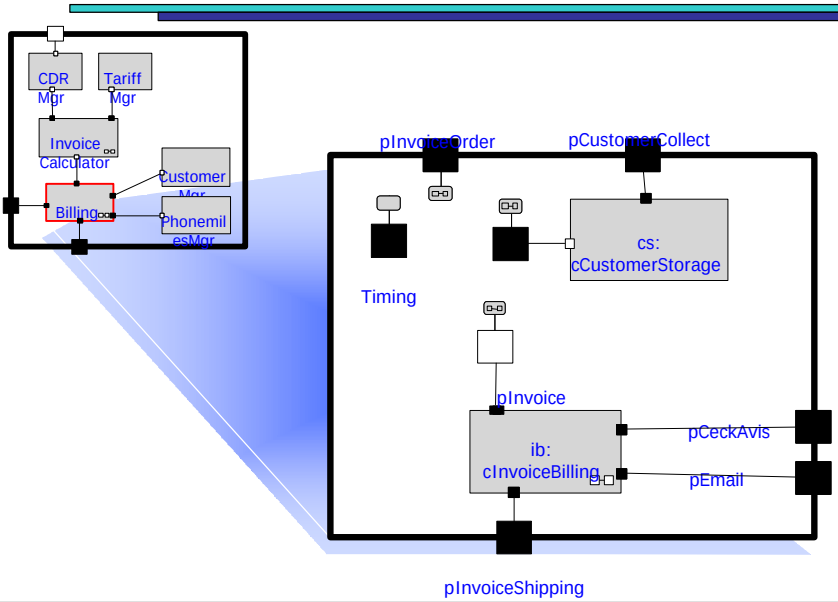


- Konnektor
  - Kommunikationsverbindung zwischen Ports
  - „fährt“ Protokoll

Struktursicht auf das Telecom-Billing-Systems



Substruktur von der Kapsel Billing

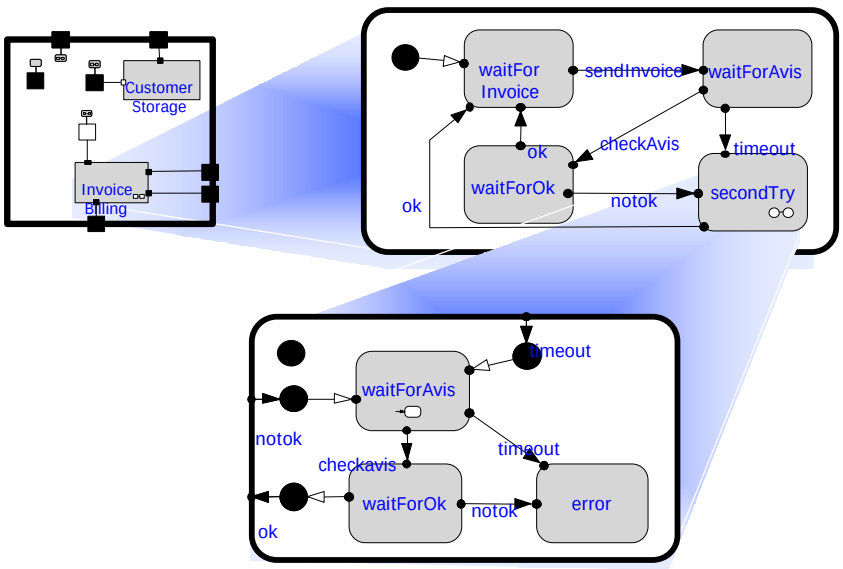


Verhaltensbeschreibung in UML-RT

- Jede Kapsel, die eigenes Verhalten besitzt, wird ein UML-statechart zugeordnet:
  - Verzicht auf AND-Zustände
  - Run-to-Completion-Semantik
- maximal ein statechart pro Kapsel
- Hierarchische Komposition:

  ⇒ jede Subkapsel kann einen eigenen Automaten besitzen

Verhaltssicht auf die Kapsel InvoiceBilling



Fazit

- Wesentliche Erweiterungen von UML-RT gegenüber UML: Modellierung von Systemarchitekturen
  - Hierarchische Kapselstrukturen
  - Transparenter Schnittstellenbegriff
  - Klares Kommunikationskonzept
  - Klares Konzept von Nebenläufigkeit
- Bewährte Konzepte der UML können genutzt werden:
  - Use-Case / Sequenzdiagramme
  - Klassendiagramme
  - Paketkonzept
  - Deploymentdiagramme

Modelling Approach according to

"Applied Software Architecture" by Christine Hofmeister et. al

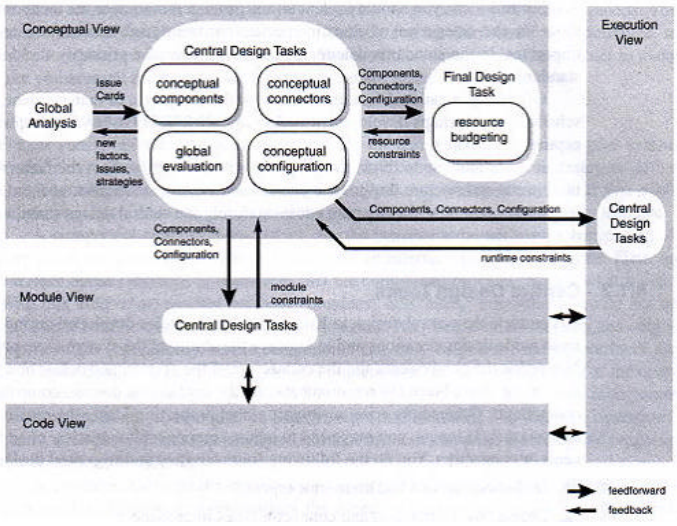
Tobias Miller - miller@4soft.de

ZEN
ADL-Workshop

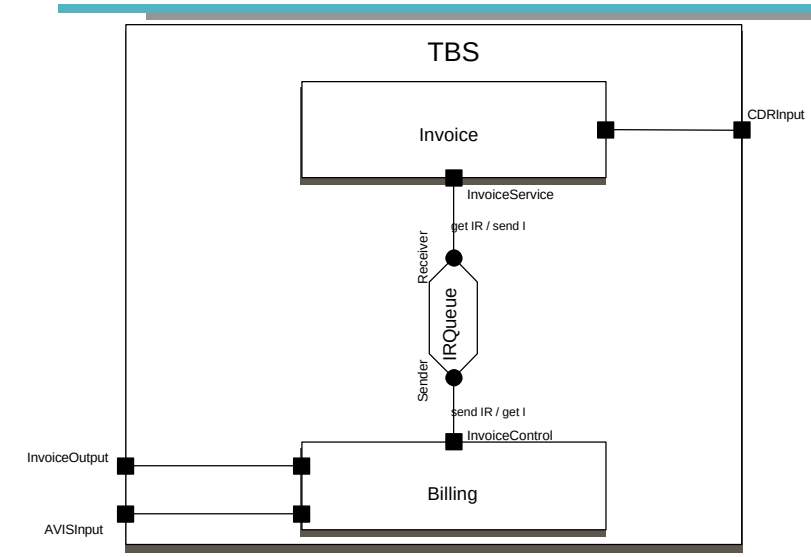
Lehrstuhl für Software & Systems Engineering
Technische Universität München

Juli 2001

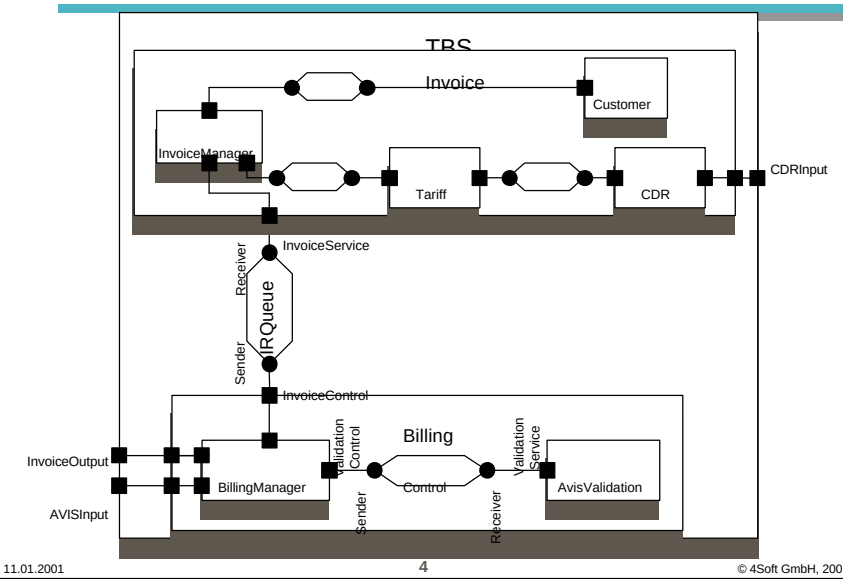
Overview



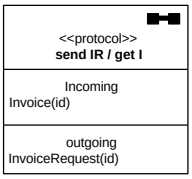
Conceptual View of the TBS

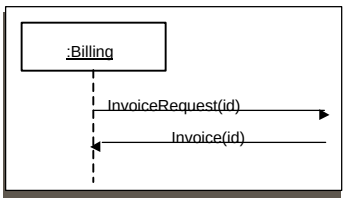


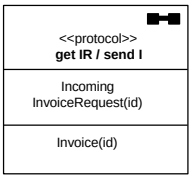
Refined Conceptual View of the TBS

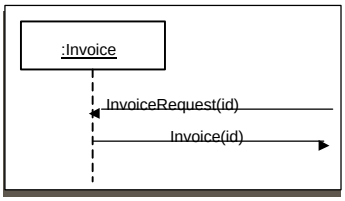


Billing / Invoice Protocol Specifications

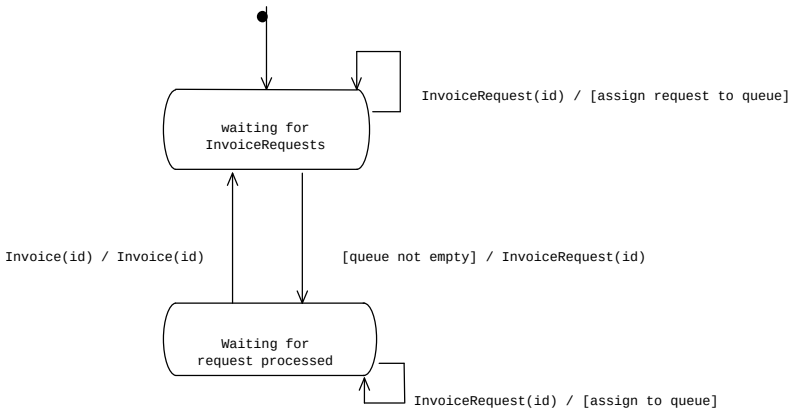




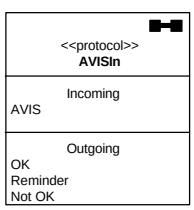


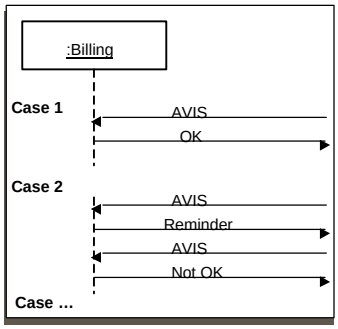


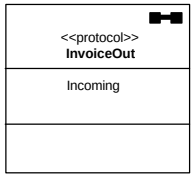
Behavior of IRQueue Connector

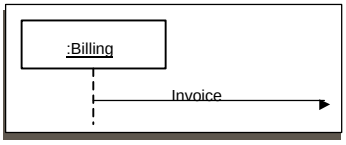


Invoice / AVIS Protocol Specifications

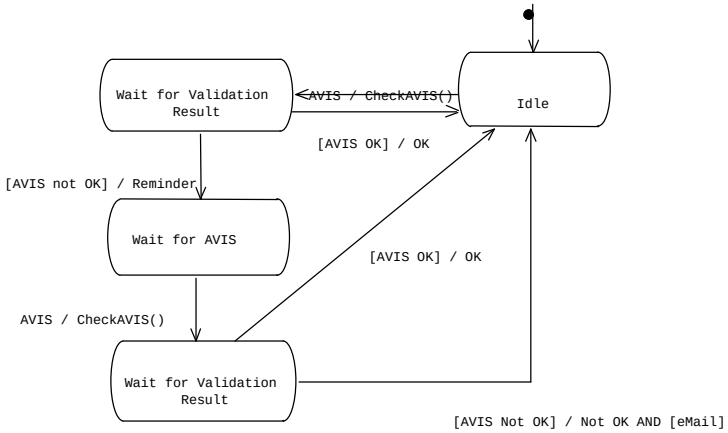








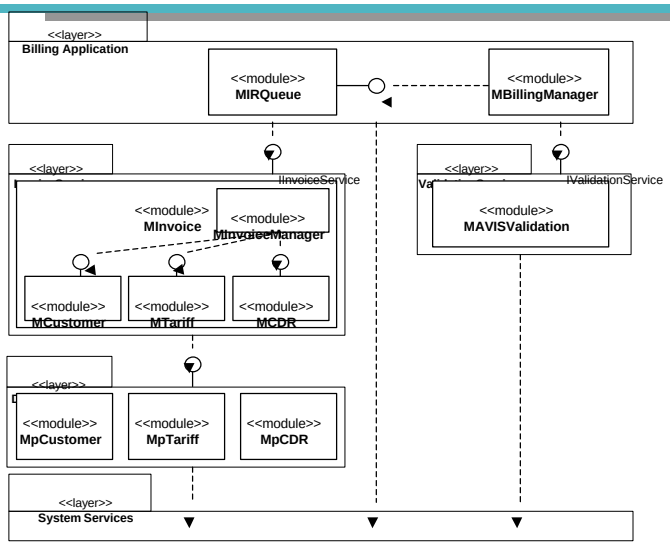
Behavior of BillingManager Component



Mapping from Components & Connectors to Modules & Subsystems

Conceptual Element		Module Element	
Name	**Kind**	**Name**	**Kind**
Invoice InvoiceService CDRInput	Component Port Port	MInvoice	Module
InvoiceManager	Component	MInvoiceManager	Module
Customer	Component	MCustomer	Module
Tariff	Component	MTariff	Module
CDR	Component	MCDR	Module
BillingManager InvoiceControl ValidationControl Control InvoiceOutput AVISInput	Component Port Port Connector Port Port	MBillingManager	Module
AVISValidation ValidationService	Component Port	MAVISValidation	Module
IRQueue	Connector	MIRQuere	Module

Assigning Modules to Layers



Execution View & Code View

- Execution View
  - Assignment of Modules to Runtime Entities
  - Identification of Communication Paths and Mechanisms
  - Description of the systems runtime topology
- Code View
  - Organisation of the systems implementation
  - Identification of source, intermediate and deployment components
  - Creation of build procedures and configuration management

Summary and Conclusions

- Supported Concepts
  - **Meta-Model:** available for each view
  - **Behaviour of architecture elements:** State- and Activitydiagrams
  - **Concepts of communication:** Sync, Async (connectors, Seq. Diagrams)
  - **Modelling of real-time requirement:** restricted to UML
  - **Concurrency:** yes, e.g. decoupled components
  - **Refinement and abstraction:** supported, but not consistent
  - **Mathematical founded:** no
- Practical Usage
  - **Practical usage:** several huge commercial systems given
  - **Graphic notation:** related to UML
  - **Architectural views:** Conceptual, Module, Execution and Code
  - **Presentability of physical distribution:** yes, via Execution View
  - **Presentability of non-functional aspects:** tables (part of the process)
  - **Process:** extended process model given
  - **High-Level Design:** yes, conceptual view

Requirements at Software Architecture Description Languages

Ingolf Krüger

Technical University of Munich
Department of Computer Science
D-80290 Munich, Germany

What is Software Architecture?

A Software-Architecture desc ition into

units / componer

together with their

connections / interacti relationships

complying to

quality attributes / development guidelines / restrictions

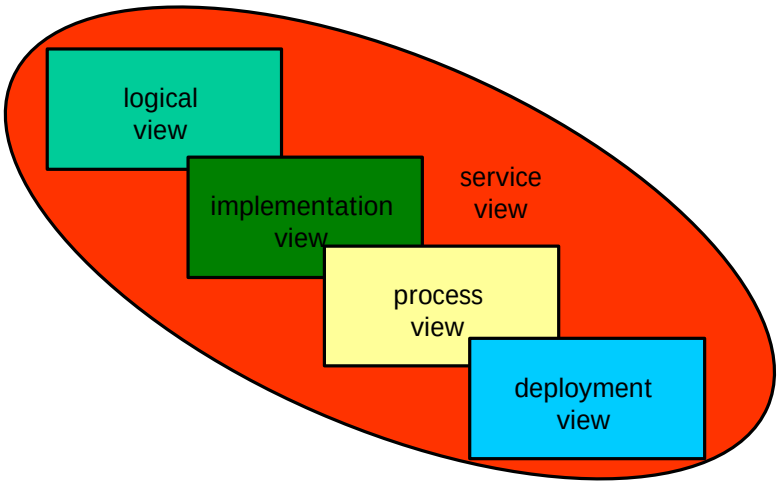
more specifically:

- interfaces (offered/used)
- behavior (at the interfaces)

(a ary) distinction bet een „functional“ und non-functional“ attributes

Aspects of Software Architecture

see also: [RUP98]



[RUP98] Philippe Kruchten: The Rational Unified Process – An Introduction. Addison-Wesley, 1998

The Role of Software Architecture

- The selection of an adequate software architecture is one of the key factors for success during system development
- Clearly structured architecture is the basis for:
  - project organization
  - complexity management
  - reuse
  - component-oriented development
  - overall system understanding, shared across team boundaries
  - manageable system evolution

⇒ Architecture description should be part of every development project

The Role of Software Architecture

A good SW-Architecture

- increases system stability
  - ⟹ small changes to the requirements result in small changes to the system
- possesses „conceptual integrity“
  - ⟹ balance
  - ⟹ simplicity
  - ⟹ elegance, clarity
  - ⟹ practicality

Modeling and Documenting Architectures

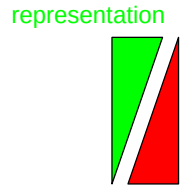
How to model and implement

- *units / components / subsystems*,
- *connections / interactions / relationships,*
- *quality attributes / development guidelines / restrictions*

adequately and systematically?

⇒ Approaches:

- *Architecture Description Languages (ADLs)*
- *architectural styles and patterns*
- *domain-specific architectures*
- *infrastructures*



Is **Your** ADL Good Enough?

Does it provide:

- a non-technical component notion
- clear concepts for hierarchy
- strong concepts and description techniques for
  - logical component distribution
  - concurrency
  - non-technical interfaces
- formal means for behavior descriptions with respect to interfaces
- support for further architectural aspects and constraints (time, performance, memory, deployment, ...)
- support for system evolution

Is **Your** ADL Good Enough?

further questions:

- is there a perfect application domain for your ADL?
- what other application domains might be of interest?
- what commonalities/differences do different ADLs face within/accross application domains?
- among the candidate ADLs of this workshop: which is the most flexible one?
- is there a perfect ADL?

Classification of Competitors

	component	connector	configuration	application domain
B-Method				
Rapide				
Focus				
? (Maurice S.)				
UML-RT				
Hofmeister et al.				
UML/Together				
Catalysis				