Two Sides of Structuring Multi-Functional Software-Intensive Systems: Function Hierarchy and Component Architecture

Manfred Broy

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany
Research Topics in Munich

• Theory of System Modelling
  ◦ Focus - Systems, Composition, Services
• Flexible Development Processes
  ◦ V-Modell XT
• Software Support Tools
  ◦ AutoFocus
  ◦ Integration tool backbones
• Verification
  ◦ Verifying real world embedded software
• Adaptive Systems
  ◦ requirements, architectures, applications
• Requirements Engineering
  ◦ Process, artefacts, models
  ◦ Product line engineering
• Integrated Modelling Software/Electronics/Hardware Systems
  ◦ Architecture
  ◦ Modelling of mechanical systems
• Software and Systems Quality
  ◦ Quality model
  ◦ Code quality management
Characteristics of today’s software intensive systems

- Interactive

- Multi-purpose functionalities
  - in cars up to 2500 functional features,
  - in mobile phones up to 700 features
  - “feature interaction”,
  - Involved man machine interfaces (multi modal access to multi-functional systems)

- Systems implemented on a network of computing nodes
  - Interactive/reactive
  - Distributed architectures (in cars up to 80 controllers, 4 bus systems and more, in airplanes 600 controllers and more)
  - Functionally connected (in cars several communication busses connected by gate ways or back bones, composed functions)
  - Networked to the environment (peer to peer communication)
Structuring Systems by Architectures

Overall Motivation

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany
Comprehensive Architecture: Levels

The structure of software-intensive systems:

- Functionality: usage view
  - Multi-functional systems: feature hierarchies
  - Feature interaction
- Logical system architecture

Conceptional Architecture

- Software Architecture
  - Design time software architecture
    - Application software
    - Software platform (OSEK, bus systems)
  - Run time software architecture
    - Tasks
    - Scheduling
- Hardware Architecture
  - Controllers
  - Communication devices
  - Sensor and actuators
- Deployment

Technical Architecture
Conceptional Architecture

• Functionality: usage/service view
  ◊ Multi-functional systems: feature hierarchies
  ◊ Functional dependencies: feature interaction

The usage view sees the system as a hierarchy of functions (features, services) that are offered by the system to its users.

• Logical system architecture: design view
  ◊ A set of components that interact by exchanging messages over channels
  ◊ By their co-operation the components generate the behaviour as modelled by the usage view (if the architecture is correct)
  ◊ Components can be further decomposed; this leads to a hierarchical component architecture

The logical system architecture defines the decomposition of the system into a set of sub-systems (the components); it constitutes the logical design.
Example: Function hierarchy (small subset)

- Body control
  - Window control
    - Child lock
    - Right back window
    - Controls
  - Locking control
    - Keyless lock
The role of the conceptional architecture in development

• Function hierarchy/service taxonomy:
The function hierarchy is to be specified in the requirements engineering
It comprises (models) all functional requirements

• Logical architecture
The has to be worked out in the design phase
It comprises the decomposition of the systems in a hierarchy of sub-systems (logical components) fixing their logical roles
A logical architecture
The comprehensive model

Usage function hierarchy
- service taxonomy

Logical architecture

conceptional architecture

Technical architecture

Software architecture

Tasks
- T1
- T2
- T3
- T4
...

Deployment

Hardware architecture

- T1 ...
- T2 ...
- T3 T4 ...

ASOSA, San Diego, June 2008  Manfred Broy
The overall goal

Provide a formal model for the comprehensive architecture and all of its views

In this talk: concentration on the conceptional architecture

• The foundation
  ◊ The basic system model: components
  ◊ System specification and verification
  ◊ System composition

• Service taxonomy

• Logical architecture

• Relationship between service taxonomy and logical architecture
The Unified Theory

Formal Foundation

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany
Towards a uniform model: Basic system model

**System class**: distributed, reactive systems

System consists of
- named components (with local state)
- named channels

driven by global, discrete clock
Timed Streams: Semantic Model for Black-Box-Behavior

**Message set:**

\[ M = \{ a, b, c, \ldots \} \]

Messages transmitted at time \( t \)

\[ <a,d,a,b> \]

Infinite channel history
The Basic Behaviour Model: Streams and Functions

C set of channels

Type: $C \rightarrow \text{TYPE}$ type assignment

$x : C \rightarrow (\mathbb{N}\{0\} \rightarrow M^*)$ channel history for messages of type $M$

$	ilde{C}$ or $IH[C]$ set of channel histories for channels in $C$
System interface model

Channel: Identifier of Type stream

\[ I = \{ x_1, x_2, \ldots \} \] set of typed input channels
\[ O = \{ y_1, y_2, \ldots \} \] set of typed output channels

Interface behavior

\[ f : \vec{I} \rightarrow \mathcal{P}(\vec{O}) \]

Set of interfaces: \( \text{IF}[I \rightarrow O] \)
System interfaces

(I ▶ O) syntactic interface with set of input channels I and of output channels O

F : \bar{I} \rightarrow \wp(\bar{O}) semantic interface for (I ▶ O) with timing property addressing causality
(let x, z \in \bar{I}, y \in \bar{O}, t \in \IN):

\[ x \downarrow t = z \downarrow t \Rightarrow \{ y \downarrow t+1 : y \in F(x) \} = \{ y \downarrow t+1 : y \in F(z) \} \]

x \downarrow t prefix of history x with t finite sequences

A system is a total behavior

Component interface

A system is a total behavior
Example: Component interface specification

A transmission component TMC

TMC

\textbf{in} \ x: T
\textbf{out} \ y: T
\textbf{x} \sim \textbf{y}

\[ x \sim y \equiv (\forall m \in T: \{m\} \odot \overline{x} = \{m\} \odot \overline{y}) \]

Specifying assertion
Composition and Decomposition of Systems

\[ F_1 \in \text{IF}[I_1 \to O_1] \]
\[ F_2 \in \text{IF}[I_2 \to O_2] \]
\[ C_1 = O_1 \cap I_2 \]
\[ C_2 = O_2 \cap I_1 \]
\[ I = I_1 \setminus C_2 \cup I_2 \setminus C_1 \]
\[ O = O_1 \setminus C_1 \cup O_2 \setminus C_2 \]

\[ F_1 \otimes F_2 \in \text{IF}[I \to O], \]

\[ (F_1 \otimes F_2).x = \{z|O: x = z|I \land z|O_1 \in F_1(z|I_1) \land z|O_2 \in F_2(z|I_2)\} \]
Interface specification composition rule

\[
\begin{array}{l}
\text{F1} \\
\text{in } x_1, z_{21}: T \\
\text{out } y_1, z_{12}: T \\
P_1 \\
\text{F2} \\
\text{in } x_2, z_{12}: T \\
\text{out } y_2, z_{21}: T \\
P_2 \\
\text{F1} \times \text{F2} \\
\text{in } x_1, x_2: T \\
\text{out } y_1, y_2: T \\
\exists z_{12}, z_{21}: P_1 \land P_2
\end{array}
\]
Composition of Specifications into Architectures

Composed component spec

\[
\begin{align*}
\text{in} & \quad x_1: M_1, x_2: M_2, \ldots \\
\text{out} & \quad y_1: N_1, y_2: N_2, \ldots \\
\exists & \quad c_1, c_2, \ldots : P_1 \land \ldots \land P_n
\end{align*}
\]

System composition = logical und

Channel Hiding = existential quantification
 STATUS OF THE THEORY

Fully worked out:
• Relationship to state machines and MSCs
• Refinement
  ◊ Property refinement
  ◊ Interaction/granularity refinement
  ◊ Time refinement
• Specification and verification calculus (implemented in Isabelle)
• Numerous examples and case studies
• Prototype tool AutoFocus

Current research
• Advanced tooling
• Product Lines (variability)
• Software/Hardware modelling
• Computability notions for concurrent interactive computations
• Application in several areas
  ◊ Automotive
  ◊ Avionics
  ◊ Automation and drive
A screen shot from AutoFocus
Service Taxonomy
Function Hierarchy

Specification of multi-functional systems

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany
Functionality and services: the users view

- Usage view onto multi-functional systems:
  - User function hierarchies (generalizing use case diagrams)
    - Taxonomy of user functions/services
    - Sub-function relation as partial order
    - Atomic functions/services as leaves in the hierarchies
  - Dependencies between user function
    - Different forms of dependencies
      - Sub-service
      - Controls
      - Uses
      - Enables
      - Is_independent
      - ...
    - Feature interaction
  - Modelling and specification of atomic functions
    - Interaction patterns given by partial functions on streams
    - Partial state machines
  - Combination of user functions into multi-functional systems
Hierarchy of usage functions ("services")
**Services/System Functions**

\[ F : \overline{I} \to \varnothing(\overline{O}) \]

service interface with the syntactic interface \((I \to O)\) with

\[ F.\mathbf{x} \neq \varnothing \neq F.\mathbf{z} \land x^t = z^t \Rightarrow \\
\{y^t+1: y \in F(x)\} = \{y^t+1: y \in F(z)\} \]

\[ \text{Dom}(F) = \{x: F.\mathbf{x} \neq \varnothing\} \]

the service domain

\[ \text{Ran}(F) = \{y \in F.\mathbf{x}: x \in \text{Dom}(F)\} \]

the service range.

\[ \text{IF}[I \to O] \]

set of service interfaces

A service may be a partial behavior
Key question

- Note:
  - A system is a component is a system

- What does it mean that:
  - A system (component) S offers a service E?
  - The projection of the system S to the syntactic interface of service E is (a refinement of) the service E!
  - A system can take a role if it offers the service defined by a role.

- Can we understand the behaviour of a multi-functional system as the hierarchy of the services it offers?

- How can we capture the dependencies between the services?
Syntactic sub-interfaces

A typed channel set $C_1$ is called a sub-type of a typed channel set $C_2$ if

- The channels in $C_1$ are a subset of the channels in $C_2$
- The messages of the channels in $C_1$ are a subset of the messages of these channels in $C_2$

We write then $C_1$ subtype $C_2$
Communication History Restriction

Given typed channel sets $C$ and $C'$ where

$C'$ subtype $C$

holds,

we denote for the channel history $x \in \text{IH}[C]$ by

$x|C' \in \text{IH}[C']$

the restriction of $x$ to the channels and messages in $C'$
Sub-types between interfaces

If for syntactic interfaces \((I_1 \rightarrowtail O_1)\) and \((I_2 \rightarrowtail O_2)\) where

\[ I_1 \text{ subtype } I_2 \]

and

\[ O_1 \text{ subtype } O_2 \]

we call the syntactic interface \((I_1 \rightarrowtail O_1)\) a sub-type of the interface \((I_2 \rightarrowtail O_2)\) and write:

\[ (I_1 \rightarrowtail O_1) \text{ subtype } (I_2 \rightarrowtail O_2) \]
Remark: Sub-types for inheritance

If for syntactic interfaces \((I_1 \rightarrow O_1)\) and \((I_2 \rightarrow O_2)\) where

\[
I_2 \text{ subtype } I_1
\]

and

\[
O_1 \text{ subtype } O_2
\]

we call the syntactic interface \((I_1 \rightarrow O_1)\) an inheritance sub-type of the interface \((I_2 \rightarrow O_2)\).

Then we can replace

\[
F_2 \in \text{IF}[I_2 \rightarrow O_2] \text{ by } F_1 \in \text{IF}[I_1 \rightarrow O_1]
\]

without running into ill-typed systems.
Faithful Projections

Given syntactic interfaces with \((I_1 \rightarrow O_1)\) subtype \((I \rightarrow O)\), we define for a behavior function \(F \in IF[I \rightarrow O]\) its \textit{projection}

\[ F^\dagger(I_1 \rightarrow O_1) \in IF[I_1 \rightarrow O_1] \]

to the syntactic interface \((I_1 \rightarrow O_1)\) by (for all \(x \in IH[I_1]\)):

\[ F^\dagger(I_1 \rightarrow O_1).x = \{y|O_1: \exists x' \in IH[I]: x = x'|I_1 \land y \in F.x'\} \]

The projection is called \textit{faithful}, if for all \(x \in \text{dom}(F)\)

\[ (F.x)|O_1 = (F^\dagger(I_1 \rightarrow O_1)).(x|I_1) \]
Variations of sub-services

Given services $F_1 \in IF[I_1 \triangleright O_1]$ and $F \in IF[I \triangleright O]$ where

$$(I_1 \triangleright O_1) \text{ subtype } (I \triangleright O)$$

holds, we call

$F_1$ unconditional sub-service of $F$, if

$$F_1 = F_1^\dagger(I_1 \triangleright O_1) \in IF[I_1 \triangleright O_1]$$

$F_1$ restricted sub-service of $F$, if there exist $R \subseteq IH[I]$ such that

$F_1$ is a sub-service of $(F|R)^\dagger(I_1 \triangleright O_1)$
What we got

• Formal notion of a **system**
  ◇ with input and output
  ◇ represented by a relation between input and output histories
  ◇ are specified by history **assertions**
  ◇ can be used as a **component** to form a large system
  ◇ can be de-composed into an **architecture** of components

• Formal notion of a **service/feature/system function**
  ◇ with input and output
  ◇ represented by a relation between input and output histories
  ◇ are specified by history **assertions**
  ◇ can be used as a **sub-service** to form a large system
  ◇ can be de-combined into an **taxonomy** of services

• **Every component**
  ◇ can be de-combined into its **taxonomy** of its sub-services
  ◇ the sub-services can be related by **service dependency relations**
Variations of dependencies

There are many variations of dependencies:

- Sub-service
- Enables/disables
- Stops
- Supports
- ...

Critical question:
Which are the significant relations
An architecture pattern

InTrans → $F_1$ → OutTrans

$F_2$
Combination of sub-services

The combination of sub-services

\[ F_1 \in \text{IF}[I_1 \triangleright O_1], \quad F_2 \in \text{IF}[I_2 \triangleright O_2] \]

into a super-service

\[ F_1 \oplus F_2 \in \text{IF}[I_1 \cup I_2 \triangleright O_1 \cup O_2] \]

such that both \( F_1 \) and \( F_2 \) are a sub-services of \( F_1 \oplus F_2 \)
Example of a service hierarchy

(from the Thesis of Sabine Rittmann)
Example of a service hierarchy

(from the Thesis of Sabine Rittmann)

Figure 4.13: Service graph of the power seat control system (service graph)
Example of a service description

(from the Thesis of Sabine Rittmann)

Figure 4.25: Standard control interface of the basic service relationship INTERRUPT (SSD + STD)
Example of a service interplay

(from the Thesis of Sabine Rittmann)

Figure 4.31: Interplay of services according to horizontal service relationship (SSD)
Combining Services/Features/Functions

Given two services $F_1$ and $F_2$ in isolation

We want to combine them into a system $F_1 \oplus F_2$
Combining Services/Features/Functions

Their isolated combination

\[ F_1 \oplus F_2 \]

\[ F_1 \]

\[ F_2 \]

\[ I_1 \]

\[ I_2 \]

\[ O_1 \]

\[ O_2 \]
Combining Services/Features/Functions

If the two services $F_1$ and $F_2$ have feature interaction we typically get:

We explain the combination $F_1 \oplus F_2$ as a refinement step.
The steps of service integration

Given the isolated service $F_1$

We construct a refinement $F'_1$

And combine $F'_1$ with a refinement $F'_2$ of $F_2$
Formal refinement

Possible formal relations between $F'_1$ and $F_1$

*faithful refinement* (maximal requirement)

$$F'_1 \uparrow (I_1 \triangleright O_1) \text{ is_service_refinement_of } F_1$$

or more sophisticated (let $F \in (I \triangleright O)$) an “existential refinement” (minimal requirement):

$$\forall x_1 \in \text{Dom}(F_1): \exists x \in \text{Dom}(F'_1):$$

$$x_1 = x|I_1 \land (F'_1.x)|O_1 = F_1.x_1$$
State of Research on Service Taxonomies

- Theory worked out and stable
  - System interface behaviour as combination of services
  - Formal foundation of use cases
- Pragmatic description techniques for function hierarchies under development
- Method for development of function hierarchies
  - Identify feature hierarchy
    - Names of services (use cases)
  - Specify features in isolation
    - By logical specifications
    - By interaction diagrams
    - By partial state machines
  - Identify dependencies
    - Use standard dependency relations
  - Specify dependencies
    - Specify dependencies by logical messages
- Case studies: Application to mobile phones and cars
Logical Architectures

The Logics of Design

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany
The role of decomposition and composition

• **Abstraction**: A system black box behaviour is described by an *interface specification*

• **Modularity**: Given a set of *components* that fit together syntactically - specified by their *interface specifications* we can *compose* them to a system (forming an *architecture*) and calculate its *interface behaviour* from the *interface behaviours* of the components
Specifications, implementation, architecture ...

- **Informal requirements**
  - Formalization

- **Requirement Engineering**
  - Validation
  - Formalized system requirements in terms of service taxonomies

- **Architecture design**
  - Architecture verification
  - S ← S1 ⊗ S2 ⊗ S3 ⊗ S4

- **Component implementation**
  - Verification
    - R1 ⇒ S1
    - R2 ⇒ S2
    - R3 ⇒ S3
    - R3 ⇒ S4

- **Integration**
  - Integration
    - R = R1 ⊗ R2 ⊗ R3 ⊗ R4
State of Research in Logical Architectures

Theory stable

• Composition
  ◊ Modular hierarchical model
  ◊ Relation to state machines (automata with input and output)

• Well-worked out theory of refinement
  ◊ Modular refinement notion
  ◊ Several notions of refinement

• Extensive interface theory

• Theory of timed and refinement
Relating Function Hierarchy and Logical Architecture

Comprehensive System Architecture

Technische Universität München
Institut für Informatik
D-80290 Munich, Germany
From the usage architecture to the logical architecture

Sub-services by projections

Logical micro-architectures by projections of the logical architecture
Tracing usage - architecture level

Each service is related to a set of logical components in the logical architecture.
RE and Product Lines

- Systematic requirements engineering is a first step into a **product line approach** (product/system families) where
  - ◊ there are classes of requirements that can be reused and
  - ◊ collections of requirements with variation points.
- Product lines are the only way in a systematic re-use
- Only with a systematic RE the step into product lines is possible
Concluding Remarks

• Today software & systems engineering is too much orientated towards the technical architecture and solutions/implementation in the beginning.

• We need a comprehensive “architectural” model-based view onto systems including requirements for dealing with complex multi-functional systems.

• The models allow for:
  ◊ Separation of concerns
  ◊ Separation technical aspects from application aspects

• Technical architectures are modelled along the same theory.

• Code can be generated from the models.
Questions ?