Inheritance in SOFA: Current State

Tomáš Opluštil

DISTRIBUTED SYSTEMS RESEARCH GROUP
http://nenya.ms.mff.cuni.cz

CHARLES UNIVERSITY PRAGUE
Faculty of Mathematics and Physics
ADL Inheritance: Practical Benefits

• Reuse
  ▪ no reinventing the wheel

• Finding definitions
  ▪ hierarchies of related definitions

• Assurance of matching definitions
  ▪ no alternative definitions

• Reading & maintaining specifications
  ▪ shorter and consistent definitions
Inheritance Techniques

• Two phases
  - Copying of ancestor elements
  - Independent modification

• Modification techniques
  - extension
    • proper
    • null
  - replacement
    • renaming
    • structural
    • cancellation
Criteria for Inheritance Mechanism Choice

• Inheritance vs. subtyping
  ▪ inheritance => subtyping => essential use
• Single vs. multiple inheritance
  ▪ name collisions
• Modification techniques
• Benefits for higher-level abstractions
  ▪ not only for inheritance
Interfaces – Example I

interface IPhoneBanking
{
    bool AcceptCall(in string passwd, ...);
    bool ProcessRequest(in long choice,...);
    void Answer(in string answerText);
    protocol: (AcceptCall; ProcessRequest; Answer)*
}

interface IGSMBanking
{
    bool AcceptSMS(in string obtainNo, ...);
    bool ProcessRequest(in string requestID, ...);
    void SendReply(in string GSMNo, ...);
    protocol: (AcceptSMS; ProcessRequest; SendReply)*
}

• Hierarchy: ProcessRequest(...) decomposed
  ▪ individual phone/gsm banking capabilities
Interfaces – Example II

```csharp
interface ICustomer
{
    bool GetCustNo(in string custID, ...);
    bool GetAccounts(in string custNo, ...);
    bool GetDisposRights(in string CustNo, ...);
    protocol: ...
}

interface IPBCustomer inherits ICustomer
{
    bool VerifyCustPasswd(in string passwd);
    protocol: ...
}

interface IGSMBCustomer inherits IPBCustomer
{
    bool VerifyCustGSMNo(in string GSMNo);
    protocol: ...
}
```
Interfaces – Inheritance Summary

• Inheritance implies subtyping
  ▪ essential use
    • inheritance hierarchies model specializations

• Single inheritance only

• Extension modification only
  ▪ proper extension
    ▪ *null-extension (possible behavior protocol spec.)*

• Benefits for higher-level
  ▪ ties in architectures
  ▪ frame subtype definition
Frames – Example I

frame PhoneBanking
{
    provides: IPhoneBanking ipB;
    requires: IPBCustomer iC;
    IAccountInformation iAccInfo;
    IAccountActions iAccA;

    protocol: ...
};

frame GSMBanking
{
    provides: IGSMBanking iGSMB;
    requires: IGSMBCustomer iC;
    IAccountInformation iAccI;
    IAccountActions iAccA;

    protocol: ...;
};
Frames – Example II

frame PhoneBankingWithInternalRequirements
{
  provides: IPhoneBanking iPB;
  protocol: ?iPB.AcceptCall;
    ?iPB.ProcessRequest; ?iPB.Answer;
};
frame PhoneBanking
  inherits PhoneBankingWithInternalRequirements
{
  requires: IPBCustomer iC;
    IAccountInformation iAccInfo;
    IAccountActions iAccA;
  protocol: ...
};

frame PassiveEBanking
  inherits PhoneBanking, GSMBanking
  renaming GSMBanking iC to iGSMCust,
      PhoneBanking iAccInfo to iAccI
{
  protocol: ... }

Tomáš Opluštil
Seminar, Charles University, Prague, 2004/03/17
Frames – Inheritance Summary

- Inheritance does not imply subtyping
- Multiple inheritance supported
  - component combination
    - design-time change of “component weight”
  - two cases of name collisions
    - solved by explicit renaming
- Modification techniques
  - extension (both proper and null)
  - renaming of interface instances
architecture APhoneBanking implements PhoneBanking
{
    inst PBCore scPBCore;
    inst SpeechProcessing scSpProc;
    delegate iPB to scCore:iPB;
    subsume scPBCore:iAccl to iAccInfo;
    subsume scPBCore:iAccA to iAccA;
    subsume scPBCore:iCustI to iC;
    bind scPBCore:iSp2Txt to scSpProc:iSpAnal;
    bind scPBCore:iTxt2Sp to scSpProc:iSpSynth;
};
architecture
APhoneBankingWLog
  implements PhoneBankingWLog
  inherits APhoneBanking
  { inst PhoneBankingLog
     sciLog;
     inst PBCoreWLog
       sciPBCrWLog;
     inst SpeechWLog sciSpWLog;
       replace sciPBCore with
         sciPBCrWLog;
       replace sciSpeech with
         sciSpLog;
       delegate iLog to
         sciLog:iLog;
       bind sciLog:iCrLog to
         sciPBCrWLog:iLog;
       bind sciLog:iSpLog to
         sciSpWLog:iLog;
  }
architecture AGSMBanking implements GSMBanking
{
  inst GSMBCore scGSMBCore;
  inst GSMCustInfoProc scCust;
  delegate iGSMB
to scGSMBCore:iGSMB;
  subsume scCust: iGSMB CustOrig
to iC;
  subsume scGSMBCore: iAccI
to IAccI;
  subsume scGSMBCore: iAccA
to IAccA;
  bind scGSMBCore: iGSMBC
  to scCust: iGSMBCustOrig;
};
frame PassiveEBanking
  inherits PhoneBanking, GSMBanking
renaming GSMBanking iC to iGSMCust,
  PhoneBanking iAccI to iAccI2,
  PhoneBanking iAccA to iAccA2
{ protocol: ...};

architecture APassiveEBanking
  implements PassiveEBanking
  inherits APhoneBanking,
    AGSMBanking
  { inst PEBCust scCst;

    replace scCust with scCst;

    replace subsume
      scPBCore:iCustI to iC
      with bind scPBCore:iCustI
        to scCst:iCPBI,
      scCst:iPBCustOrig
        to iC;
  }
Architecture Inheritance Summary

• Inheritance does not imply subtyping

• Multiple inheritance supported
  • “coordinated” frame and architecture inheritance:
    \[ CTa_1=(Fa_1, Aa_1), \; CTa_2=(Fa_2, Aa_2) \Rightarrow \]
    \[ CTD= (Fd, Ad) \text{ where } Fd \text{ \text{inhFR} } Fa_1, Fa_2 \text{ and } Ad \text{ \text{inhARCH} } A_1, A_2 \]

• Modification techniques
  • extension (proper, null)
  • replacement
    • renaming of subcomponents
    • of subcomponents for extension (except of interface renaming)
    • transitive replacement of ties
    • renaming of ties
    • of ties for the same ties but using different connector
architecture ArchName implements Frame
  inherits AncArch, ...
  renaming AncArch::scompNm to scompNewNm, ...
  
  { inst Frame aScompNm; ...
    replace ancScompNm with anotherScNm
    using (newIntI1, ... , newIntIN)
    instead of (origIntI1, ..., origIntIN); ...
    replace origTie with aNewTiel, aNewTie2, ...;
    anotherNewTie; ...
  }
Prototype Implementation Overview

- Problem – patching existing implementation

- Java prototype CDL compiler – 2 phases
  - internal tree (SOFA SOFAnode Made CDL)
  - transformation to TIR remote objects (SOFA SOFAnode Made TIR)

- Already done:
  - compiler for interfaces, frames, (almost) architectures and interface protocols
  - initial representation in TIR
  - compilation of definition referring to definitions in TIR
Implementation – Abstraction Elements

- Container-contained relationship
- Frames
  - Provides
  - Requires
  - Property
- Architectures
  - Instances
  - Bind (2x BindOper (->BindType) BindKind, mode, using)
  - Property
• Interfaces restricted to single inheritance
• F & A with inheritance = new *objectkinds*
  ▪ as descendants of original abstractions
• CompFrameInh
  ▪ FrameAnc, FrameAllAnc, FrameRen, FlatFrame
• CompArchitectureInh
  ▪ ArchAnc, ArchAllAnc, scRen, scReps, TieReps, FlatArchitecture
• “toNormal”
Implementation – TIR Phase

• No new definitionkinds
  ▪ abstractions with inheritance unfolded
  ▪ enhanced with inheritance information
    • isDescendant()
    • getInheritanceStructure()

• Fully compatible, new TIR
  ▪ reads old definitions
  ▪ looks like old TIR for subsequent processes
Implementation - Problems

• Too complicated SOFA implementation!
  ▪ bindings – CDL: implicit; implementation: explicit
  ▪ completely different compiler and TIR impl.

• Versioning
  ▪ based on profiles
  ▪ version specification checking
  ▪ elements of different version in a single container

• TIR implementation strategy
  ▪ time and compatibility over space-efficiency
Evaluation

• Different approach from OOPLs
  ▪ no implementation, no notion of objects
  ▪ => no delegation, no life-time sharing

• Different approach from ADLs
  ▪ inheritance for all involved abstractions
  ▪ inheritance of components’ abstractions “coordinated”
  ▪ accidental use of inheritance
  ▪ attempt to

• Architecture inheritance maxim. expressive power
  ▪ syntax transparency
  ▪ behavior-protocol compliance problems
Open Issues, Future Work

- Implementation problems above
- Inheritance & behavior protocols
  - restricting inheritance
  - inheritance of protocols
- Implicit connector generation
- Code & CDD generation
- Applicability in different ADLs
  - control interfaces
  - non-functional properties, …