ソフトウェアシステムの高信頼かつ効率的な開発を可能にするために

～実用的な仕様記述言語の設計と、コンポネントベース開発における仕様記述の応用～

本位田研究室： 井上 拓, ニーストロム・ヨーハン, 日野 克重
Honiden Laboratory: Taku INOUE, Johan Nyström, Katsushige HINO

どんな研究？

ソフトウェア工学の以下の技術分野を軸として、様々なアプローチから研究を行っています.

- ・ソフトウェア仕様の形式的な記述
- ・コンポーネントベース開発

何が出来るの？

ソフトウェアの形式的な仕様記述を, 自然に行うことが出来ます. また現在人手に頼っているコンポーネントの合成を, 半自動で, より安全に行うことが出来ます.

10年後の姿は？

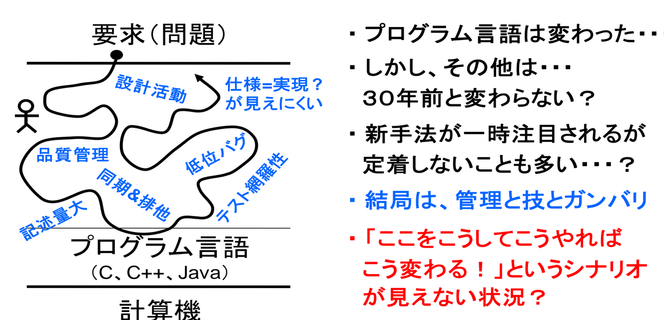
我々の研究成果を適用することで, 社会で使われる実用ソフトウェアの高信頼かつ効率的な開発を可能にすることを目指しています.

ソフトウェアの問題記述のための形式化自然語

FNL：Formalized Natural Language

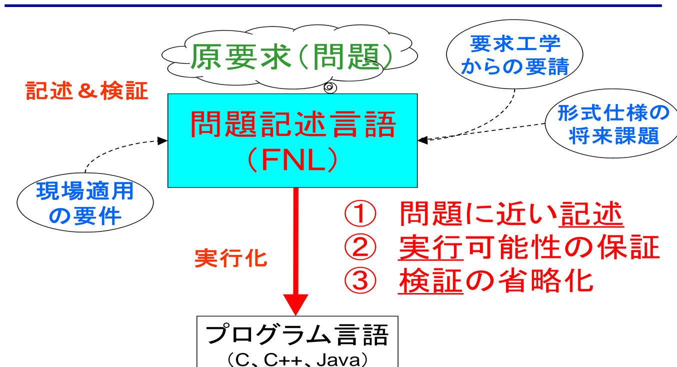
1．研究の目標 （問題＆動機）

ソフト開発現場の現状



- ・プログラム言語は変わった・・・
- ・しかし、その他は・・・ 30年前と変わらない？
- ・新手法が一時注目されるが定着しないことも多い・・・？
- ・結局は、管理と技とガンバリ
- ・「ここをこうしてこうやればこう変わる！」というシナリオが見えない状況？

2．解答案



3．言語FNL：設計方針

一般技術者が使える仕様言語！

（1） 記述性　　FNL：Formalized Natural Language
- （a）自然な存在論　（b）自然語指向
- （c）推論性　（d）統合性（静+動+並）
- （e）素朴＆直接性　（f）問題純粋性

（2） 形式性
- （g）形式意味論を備えていること

（3） 実行性
- （h）プログラム導出orその裏付け

⇒ これらを最大限に満足する言語を追究

4．言語FNL：基本詞

品詞	基本詞
定義詞	==
叙述詞	is becomes does
普通名詞	Thing Number String Set （任意）
実在詞	existent nil
属性詞	（任意）
関係詞	（任意）
事態詞	true false （任意）
行為詞	（任意）

品詞	基本詞
論理詞	not and or if
限定詞	all some many the
主体詞	@
時相詞	=> while when
修飾詞	where who which whose
助詞	: 's for

その他、算術詞、数詞、固有名詞、指示詞など

5．言語FNL：記述例 （座席予約）

```
 1: Reserve (the Member:m, the Seat:s) ==
 2:    ( s becomes reserved-by m ) or
 3:    ( does not anything ).
 4: Cancel ( the Seat:s ) ==
 5:    s becomes not reserved-by any Member.
 6: Query ( the Seat:s ) ==
 7:    does list some Member:m,
 8:       where ( s is reserved-by m ).
 9: true ==
10:    for all Seat:s
11:      { not ( s is reserved-by
12:                          many(2) Member ) }.
```

ビジネスコンポーネント間に跨る参照関係の抽出

Background／Problem

ビジネスソフトウェアの特徴：データの蓄積・活用
例：CRM、購買システム、人事システムなど

大規模なシステムはコンポーネントベース開発[1]が行われる
1) 部品（コンポーネント）を組み合わせてシステムを構築する開発手法. コンポーネントはインターフェースを介して結合される.

ビジネスコンポーネント間に跨る参照関係[2]が存在
2) 各コンポーネントに局在する論理的な永続データ間の依存関係. モデル図面からは, その存在が読み取れない.

⇒メンテナンス・追加開発時に, 参照不整合が発生する恐れがある.

<<component>> Component1 / <<component>> Component(new) / インターフェース / <<component>> Component2 (Data2, name:String) / 論理データ / <<component>> Component3 (Data3, id:String) / ?? / Destructive access / Reference might be broken!

Solution

コンポーネント仕様・設計モデルのデータフロー解析

コンポーネント操作内の代入関係を解析

```
Context Component3::operation3(name:String)
Post: Data3.allInstances()->includes(i| i.id = name)
```

操作の事後条件（OCL[3]）
3) 形式的な仕様記述言語. UMLモデルの制約を記述することができる.

コンポーネント間の操作の呼び出し関係を解析

:component1 / :component2 / :component3 / operation() / name=operation2() / operation3(name)

UML[4]のシーケンス図
4) オブジェクトモデリング言語

設計段階で参照関係を把握することができる
整合性の取れたモデルを設計し, 安全なシステムを構築することが可能になる.

<<component>> Component1 / <<component>> Component2 (Data2, name:String) / <<component>> Component3 (Data3, id:String) / Semantic reference

Poplar: State based integration

Application code → Poplar layer / Poplar layer → Interface / Interface → GUI library / JDBC library → Java libraries

The integration concern

Almost all software is composed of multiple components that must interact.
"Client" components usually have quite a lot of knowledge about "service" components. This knowledge makes further evolution of both the client and the service difficult and places a strain on the development process.
In the "Poplar" project, we introduce an auto-generated layer to minimise the shared knowledge between clients and services. With such a layer, client and service components can evolve more independently.

Typestate

By defining protocols/automata for each type, we can distinguish between valid and invalid interaction sequences, such as "new, deposit" in this figure.

commit / deposit / withdraw / open Transaction / ready / new / beginTransaction

- •new, beginTransaction, commit
- •new, beginTransaction, beginTransaction
- •new, beginTransaction, deposit, withdraw, commit
- •new, deposit
- •new, beginTransaction, deposit, beginTransaction

Partial order planning (POP)

Planning algorithms identify a sequence of actions that can realise a given goal state from a given starting state. Planning is a well studied problem in artificial intelligence, but has not yet been widely applied in software engineering. We use the Partial Order Planning (POP) algorithm, which searches the space of all possible plans until a plan has been found.

<start> (A, B) / Action 1 (A; Not-A, C) / Action 2 (B; D) / Action 3 (C, D; A) / <finish> (A, D)

Towards highly flexible software integration

We layer several automata on top of each other in a custom specification language and use POP to generate valid integration code. Currently we provide *semiautomatic* integration of software components that must be inspected by programmers. The long term goal is *fully automatic* component integration that considers the *context* of the integration.

PSL Specifications / PQL Queries / Build env. Classpath, jars, etc → Planner → Solution (code)

Component pool: C1, C7, C2, C8, C3, C4, C5, C6
Semiautomatic integration →
Correctly composed system: C2 → C3, C4; C4 → C8

NII 国立情報学研究所 National Institute of Informatics
連絡先： 本位田 真一 (Shinichi HONIDEN) 国立情報学研究所 本位田研究室
TEL: 03-4212-2513, FAX: 03-3556-1916, Email: honiden@nii.ac.jp