

実用性が高い形式工学手法 と支援ツールの研究開発

法政大学・大学院・情報科学研究科

劉少英

Email: sliu@hosei.ac.jp

HP: <http://cis.k.hosei.ac.jp/~sliu/>

報告目次

1. ソフトウェア開発の問題点
2. 改善策の提案と予想効果
3. 実用性が高いSOFL形式工学手法
4. 支援ツールの研究開発
5. SOFL形式工学手法の応用事例
6. 企業へ導入の提案
7. まとめ

1. ソフトウェア開発の問題点



管理者

何故プロジェクトはこれほどコストが高く、時間がかかるのか？

顧客

何故私が要求した通りにソフトウェアが動かないのか？



開発者

何故バグはこれほど沢山あるのか？

何故自分が書いたコードでも分からなくなるのか？

2. 改善策の提案と予想効果

実用性が高いSOFL形式工学手法

(Structured Object-oriented Formal Language)

要求仕様と設計仕様を
三段階形式仕様記述技術
によりしっかり完成させる。

仕様アニメーションにより
顧客と開発者のコミュニ
ケーションを強化する。

↑
支援ツール

予想効果

- (1) ソフトウェアテストと保守の**コスト**を、約N%以上削減することが可能, $N > 0$.
- (2) プロジェクト全体の**時間**は、約M%以上削減することが可能, $M > 0$.

SOFL形式工学手法の特徴

簡易性

可視性

明確性

対話性

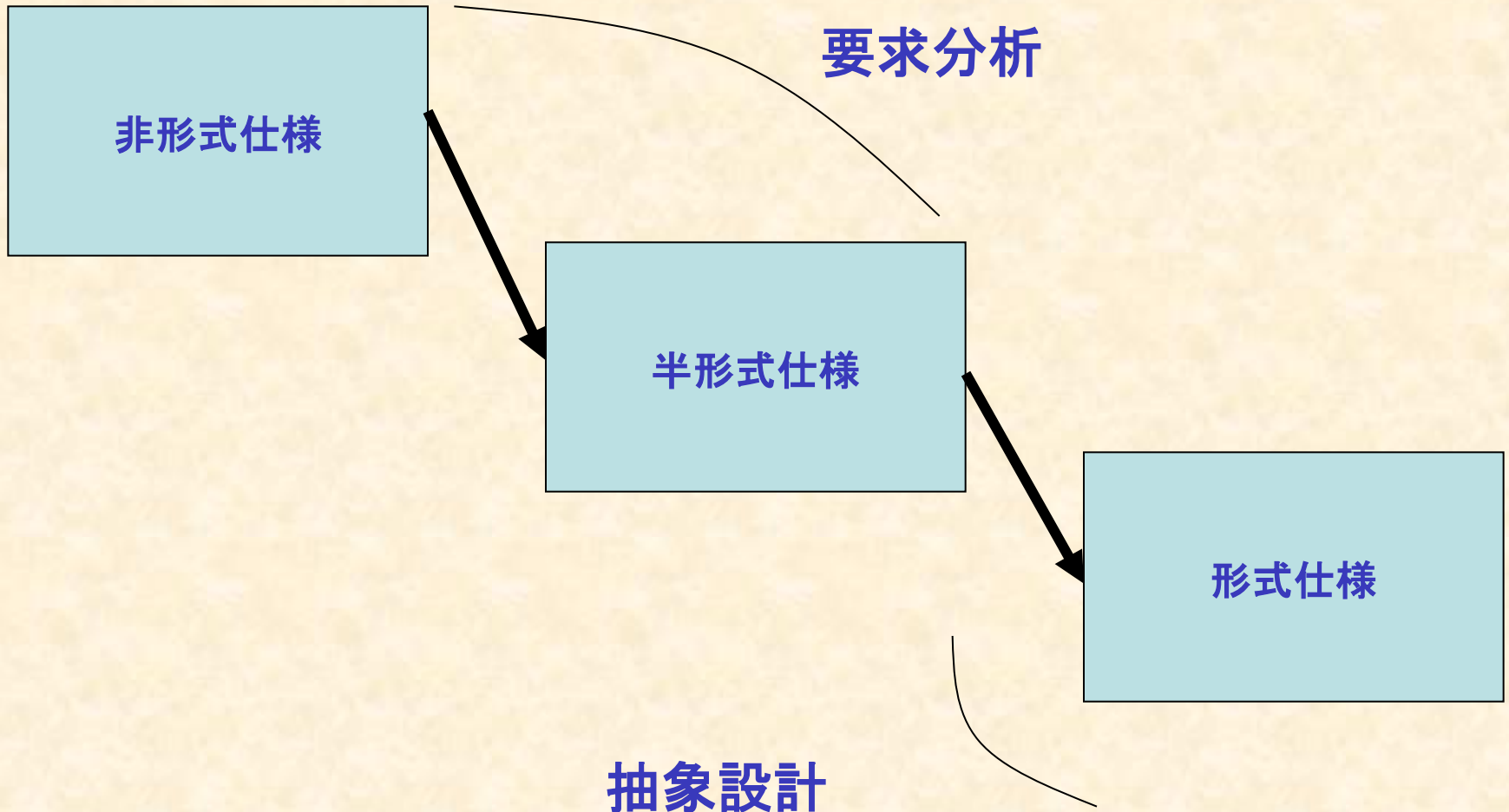
3. 実用性が高いSOFL形式工学手法

SOFL三段階形式仕様記述技術

仕様アニメーション技術

仕様パターンに基づく形式仕様の作成支援技術

SOFL三段階形式仕様記述技術



事例：JR東日本のSuicaカード

Functions

0. An IC card system for buying tickets or charging the card.
 1. Receive a service selected by a customer.
 2. Receive a card (railway pass).
 3. Buy ticket
 - 3.1 Check the card buffer.
 - 3.2 Update the card buffer.
 - 3.3 Issue a ticket.
 4. Charge card with cash.
 5. Update fixed term railway pass with cash.

Data resources

1. card buffer
2. railway pass status (including various fixed-term services)
3. tickets_available

Constraints

1. The maximum amount of the card buffer is 100,000 JPY.

非形式仕様（要求仕様）:

1. 機能

2. データリソース

3. 制約

```

module Purchase_Ticket_and_Charge_Card;
type
  Card = composed of
    buffer: nat0
    railpass_status: RailPassStatus
  end;
  RailPassStatus = composed of
    status: {<Ordinary>, <Student>, <Senior>}
    period: {<3 Months>, <6 Months>, <12 Months>}
  end;
  ServiceRequest = composed of
    Service_choice: {<Buy Ticket>, <Charge Card>,
    <Update Railpass>}
    money_amount: nat0
  End;
  Ticket = composed of
    number: nat0 /*unique number for each ticket*/
    value: nat
  end;
var
  card: Card;
  ticketPool: set of Ticket;
inv
  The buffer of card should be less than or equal to 100,000 JPY.

process Receive_Card(Inserted_card: Card )
  ext wr card: Card
  pre true
  post assing the information of the inserted card to the state variable
  card for the other processes in this module to use.
end_process;
process Buy_Ticket(ticket_price: nat)
  Ext wr card: Card
  pre true
  post (1) check whether the ticket_price is less than the card buffer.
  (2) if (1) is true, a ticket can be bought, and the card buffer is
  updated.
  (3) if (1) is false, then an error message is issued.
end_process;
process Charge_Card(amount: nat )
  ext wr card: Card
  pre true
  post If the addition of the existing amount of the card buffer and the
  input amount for charging the card is greater than 100,000 JPY,
  then an error message of exceeding the limit is issued; otherwise,
  add the input amount to the card buffer.
end_process;
process Update_RailPass(term_status: RailPassStatus, cash: nat)
  ext wr card: Card
  ...
end_process;
process Service_Selection(service_req: ServiceRequest)
  ...
end_process;
end_module;

```

半形式仕様(要求仕様):

モジュールの集合

一つのモジュールには、

(1) 必要なデータ項目

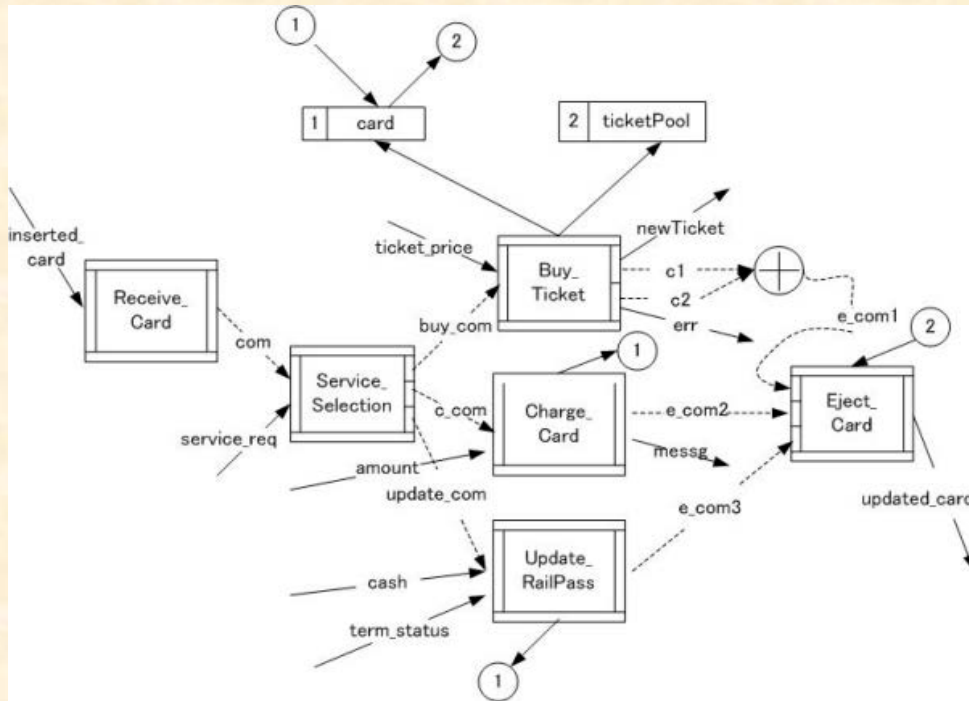
(2) データ型

(3) 操作の機能

を定義している。

形式仕様(設計仕様):

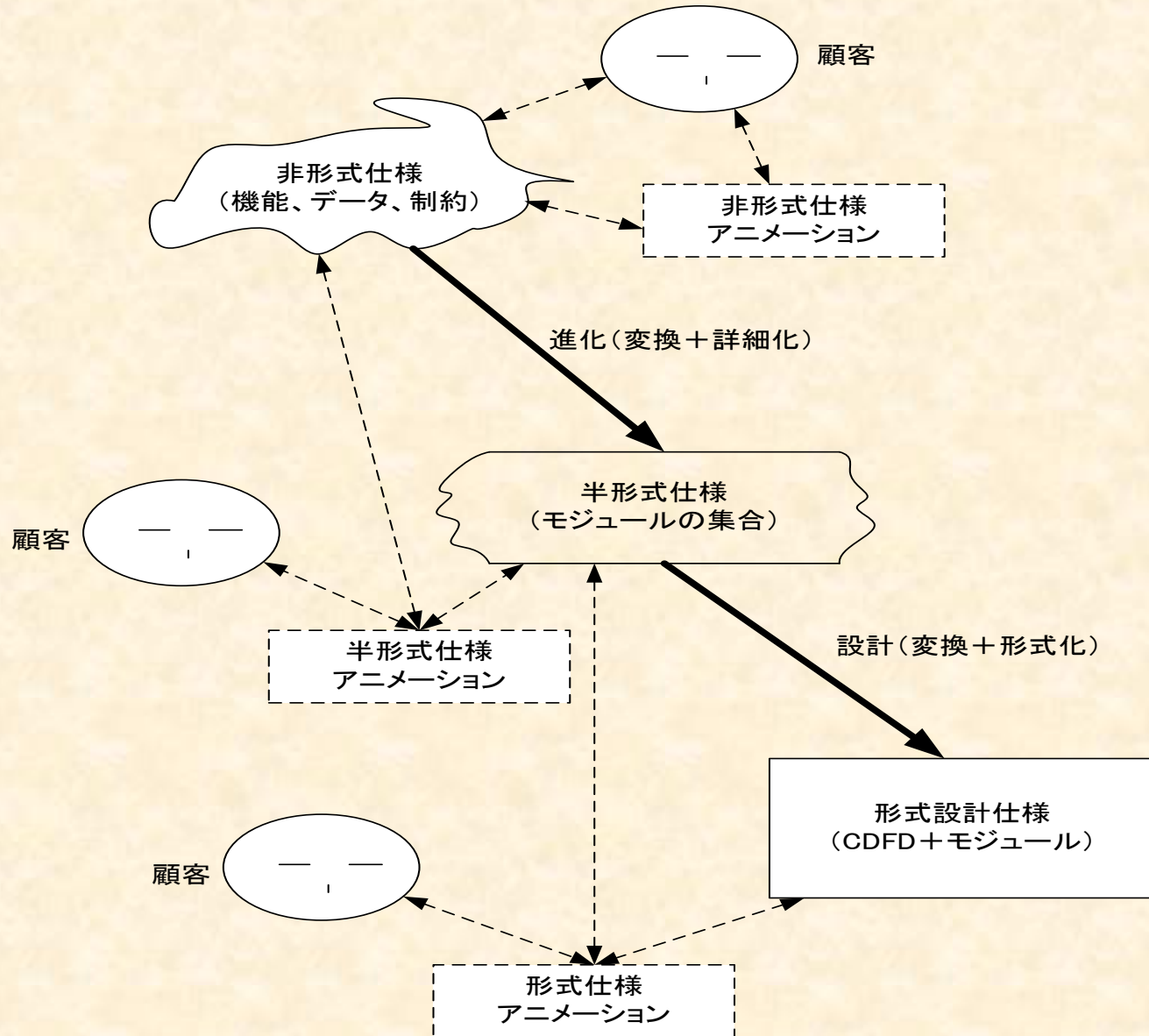
CDFD + モジュール



```
module Purchase_Ticket_and_Charge_Card;
type
  ... /*The same as that of the semi-formal specification */
var
  card: Card;
  ticketPool: set of Ticket;
inv
  card.buffer <= 100000; /* Formalization of the informal invariant*/

process Receive_Card(inserted_card: Card ) com: sign
  ext wr card: Card
  pre true
  post card = inserted_card
end_process;
Process Buy_Ticket(ticket_price: nat, buy_com: sign)
  newTicket: Ticket, c1: sign | c2: sign, err: string
  ext wr card: Card
  wr ticketPool: set of Ticket
  pre true
  post if ticket_price <= ~card.buffer
    then newTicket = get(ticketPool) and
         newTicket.price = ticket_price and
         card = modify(~card, buffer -> ~card.buffer - price) and
         ticketPool = diff(~ticketPool, [newTicket])
    else err = "Your card has no sufficient money for the ticket."
  end_process;
process Charge_Card(c_com: sign, amount: nat )
  e_com2: sign, messg: string
  ext wr card: Card
  pre true
  post if ~card.buffer + amount <= 100000
    then card = modify(~card, buffer -> ~card.buffer + amount) and
         messg = "the charge is successful."
    else messg = "your amount is over the limit."
  end_process;
process Update_RailPass(update_com: sign, cash: nat, term_status: RailPassStatus)
  e_com3: sign
  ext wr railpass_status : RailPassStatus
  ...
end_process;
process Service_Selection(com: sign, service_req: ServiceRequest)
  ...
end_process;
end_module;
```

仕様アニメーションの適用



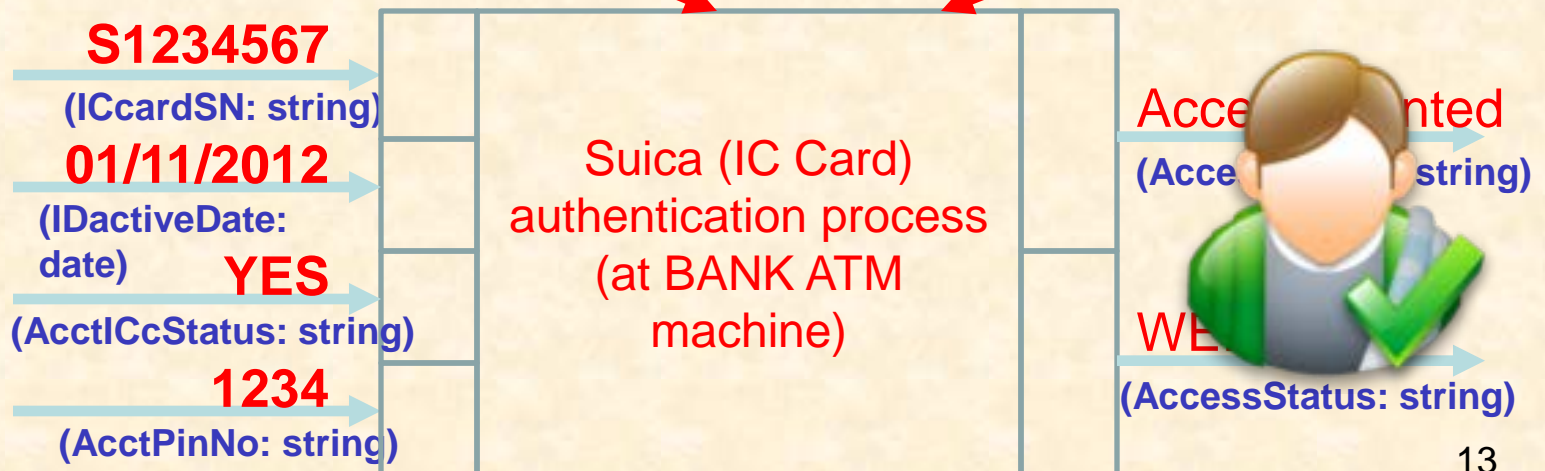
Suica authentication process

(at Bank ATM Machine)

Input (IN)				Invariants (INV)	Output (OT)			
variable	Data type	Sample Data	Data Store		variable	Data type	Sample Data	Data Store
ICcardSN	string	S1234567	D1: ICcCurrent_Detail		permission	sign	Access granted	-
IDactivateDate	date	01/11/2012	D1: ICcCurrent_Detail		elertmsg	string	WELCOME	-
AcctICcStatus	string	YES	D2:ICc_BankAcct		elertmsg	string	Invalid PIN No	
AcctPinNo	string	1234	D2:ICc_BankAcct					

D1	ICcCurrent_Detail
----	-------------------

D2	D2:ICc_BankAcct
----	-----------------



4. 支援ツールの研究開発

The screenshot displays the SOFL (Software-Oriented Formal Language) environment. The main window shows a state transition diagram for the `SuicaCard.cdfd` model. The diagram consists of several states: `Init`, `Update`, `payment`, and `Reference`. `Init` receives `suicaData` and outputs `errorMessage` and `success`. `Update` receives `buffer`, `user`, `commuter`, and `bank`, and outputs `errorMes` and `success`. `payment` receives `payment` and outputs `errorMessage` and `success`. `Reference` receives `ref` and outputs `suicaData`. Transitions connect `Init` to `Update`, `payment` to `Reference`, and `Reference` to `Init`. Additionally, there are two data stores: `2 bank_acc` and `1 card_info`, with arrows indicating data flow between the states and these stores.

The right-hand pane shows the type declarations for the `SuicaCardModule*`. The declarations are as follows:

```
Type Declaration
type
SuicaCard = composed of
  user_info:User
  buffer:Buffer
  commuter_pass_Info:CommuterPass
  bank_info:BankInfo
end
User = composed of
  name:String
  address:string
  date: string
end;
Buffer = composed of
  money:nat0
end;
module SuicaCard/
const
  minimum_require = 130;
type
SuicaCard = composed of
  user_info:User
  buffer:Buffer
  commuter_pass_Info:CommuterPass
  bank_info:BankInfo
end
User = composed of
  name:String
  address:string
  date: string
end;
Buffer = composed of
  money:nat0
  use_history : seq of string
  use_kind : seq of string
end;
CommuterPass = composed of
  start_station :string
  start_line : string
  end_station : string
  end_line : string
  limitation : {<0>,<1>,<3>,<6>}
  start_time : Date
end;
BankInfo = composed of
  bank_name : string
  branch_name : string
  account_number : string
```

5. SOFL形式工学手法の応用事例

- ① 単純化されたJR東日本のSuicaカードシステムの要求仕様と設計仕様
- ② 旅行会社システムの要求仕様と設計仕様
- ③ スマート交通信号システムの要求仕様と設計仕様
- ④ 鉄道踏切制御システムの設計仕様
- ⑤ ATMシステムの要求仕様、設計仕様、および実装
- ⑥ 郵便荷物の配達システムの要求仕様、設計仕様、および実装

6. 企業へ導入の提案

試験により導入かどうかを決める.

試験の実施方法:

- (1) 会社から2、3人の開発者を選ぶ.
- (2) SOFL形式工学手法を約5日間勉強する
(毎日約6時間).
- (3) 会社の実際の開発プロジェクトの約5%を、
SOFL形式工学手法で開発する.
- (4) 上記(3)の結果により、会社に導入するか
どうかを判断する.

7. まとめ

- (1) 実用性が高いSOFL形式工学手法は、企業のソフトウェアおよびシステム開発に非常に役に立つ。
- (2) SOFL形式工学手法は、ソフトウェア開発のコストとソフトウェア信頼性を大幅に改善できる。

ご清聴ありがとうございます！