修 士 論 文

形式手法による短距離無線通信規格の検証に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

高村 純平

2008 年 3 月

修 士 論 文

形式手法による短距離無線通信規格の検証に関する研究

指導教官　二木厚吉 教授

審査委員主査　二木厚吉 教授
審査委員　Rene VESTERGAARD 准教授
審査委員　緒方和博 特任准教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

610054 高村 純平

提出年月: 2008 年 2 月

概 要

本稿では、2003 年 12 月に ISO/IEC IS 18092 として国際基準となった短距離無線通信規格 NFC の仕様を形式手法を用いて形式化し，その安全性，信頼性を明らかにする．具体的には国際標準規格 NFCIP の仕様を OTS/CafeOBJ 法により記述を行い，形式的に検証を行う．もし，仕様に問題があれば仕様の改善点等を提案する．これにより，NFC に基づいたシステムの安全性，信頼性を高めることができると期待される．

目 次

第1章　序論

近年，利便性の点で優れているICタグ[1]が利用されている．その利用は駅の改札から入退室管理システム等多岐にわたっており，現在なくてはならないものとなってきている．ICタグはRFID[2][3][4]の技術を利用したデバイスで，ソニーが開発したFeliCa[5]や，欧米で広く利用されているMifareがその代表例である．そして現在FeliCaや，Mifareの普及により，RFID[2][3][4]の技術が注目されている．RFIDはID情報を埋め込んだタグから，無線通信によって情報をやりとりする技術全般を指す[6]．NFC(Near Field Communication)[7]はソニーとNXP(旧フィリップス)が共同で開発した無線規格で，2003年12月に国際標準規格となった．NFCはFelica，Mifare等と通信互換性があり，様々な分野での応用を期待されている．一方，近年ソフトウェア開発は，巨大化複雑化が進み，さらに短期間で開発されているケースが多い．そういったことがあって，仕様記述や信頼性の確保が課題となっている．そこで信頼性確保のために形式手法が注目されている．形式手法は，文法や意味を厳密な数学モデルや論理体系に基づいて仕様記述，検証を行う手法で，自然言語や図，表などの曖昧な表現をなくすことができる．ソフトウェア開発の上流工程で作成されるソフトウェアの仕様を形式手法により記述，検証を行うことによって，ソフトウェアのミスが早期に発見でき，開発をスムーズに行うことができる．また，それにより，信頼性の向上が期待できる．形式仕様記述言語としてはさまざまなものがあるが，本稿ではCafeOBJ[8][9]を用いる．CafeOBJは通信プロトコルなどのシステムの仕様記述や検証に用いられ，実際に成果を上げている[10]．

本研究では，NFCの仕様を形式手法，具体的には代数仕様記述言語CafeOBJを用いて，OTS/CafeOBJ法[11]により，記述・検証を行うことによって，NFCの信頼性を明らかにする．また，検証により問題が発見できれば，仕様の改善によって，その問題を解決する．これにより，NFCに基づいたシステムの安全性，信頼性を高めることができる．

以下，第2章で，NFCの安全性，信頼性を評価するために本稿で扱う手法，形式手法について述べる．また，NFCの形式的な記述に用いた代数仕様記述言語CafeOBJについても述べる．第3章で，本稿で，代数仕様記述言語CafeOBJを用いて形式的に記述，検証を行う対象となった短距離無線通信規格NFCについて述べる．第4章で，システムのモデル化，検証のための方法である，OTS/CafeOBJ法について述べ，対象となるNFCをOTS/CafeOBJ法を用いて，どのように形式的記述を行ったかについて述べる．また作成した仕様についても述べる．第5章で，作成した仕様について，NFCに期待されている性質を定式化し，その性質が満たされているかどうかについて検証を行う．第6章で，まとめと今後の課題を述べる．

第2章　形式手法

この章では，仕様の安全性，信頼性を確かめるために本研究で行う手法，形式手法について述べる．また仕様の検証に用いた代数仕様記述言語 CafeOBJ についても述べる．

2.1　形式仕様

ソフトウェアのバグの発生は，開発の上流工程に原因があることが多い．間違いがより上流の工程で存在すればするほど，修正により多くの労力が費やされる．よって，ソフトウェア開発において仕様を作成する段階が極めて重要になる．これまでソフトウェアの仕様は自然言語や図，表で表現されることが多かったが，自然言語は曖昧で客観性に乏しいので，自然言語を用いて矛盾のない厳密な仕様を書くのは難しい．これらの問題を解決するために考案されたのが形式仕様という考え方である．形式仕様とは，仕様記述の段階で，文法・意味を厳密な数学モデルや論理体系に基づいて定義された形式仕様言語を用いて表現された仕様のことである．形式仕様言語で記述された仕様は，仕様自体の機械処理が可能であり，上流工程の時点で機械的な解析・検証も可能であるので，より信頼性の高い仕様を作成することができる．形式的仕様記述言語には，状態モデルに基づくものと代数的仕様に基づくものがある．状態モデルに基づく仕様記述言語では，システムの性質をシステムの状態と操作に基づく，不変式と事前事後条件によって定義する．代数的仕様記述言語では，システムを相互に通信するプロセスの集まりとしてモデル化し，抽象データ型ごとにそれを操作する関数仕様を方程式で記述する．ここでは代数的記述仕様記述言語 CafeOBJ について述べる．

2.2　代数仕様記述言語 CafeOBJ

CafeOBJ は代数仕様言語に分類される形式仕様言語であり，仕様を構成する等式を書換規則と解釈して実行することが出来る．等式の実行は仕様の意味を想定する等式論理に忠実であるので CafeOBJ システムを用いた対話的検証が可能である．CafeOBJ は，主に，抽象データ型を記述するための始代数及び抽象機械，あるいはオブジェクト指向におけるオブジェクトを記述するための隠蔽代数に基礎をおいている．このため，抽象データ型と抽象機械を統一の枠組み，つまり代数で記述することを可能とする．

2.2.1 構文

CafeOBJ の仕様はモジュール単位で記述される．モジュールはシグネチャと公理により定義され，シグネチャはソートと演算子により定義され，公理はシグネチャによって作られる 2 項間の等式によって定義される．以下，CafeOBJ の各構文について簡単に説明する．

モジュール

CafeOBJ で仕様を記述するには，まずモジュールを宣言し，その中に様々な宣言をする．モジュールは”mod”を使って次のように宣言する．

mod *module_name* {
  *module_element**
}

module_name にはモジュール名を指定し，*module_element* には演算や等式の宣言などが記述される．

ソート

代数において，ソートは (sort) はプログラミング言語の型 (type) に対応する概念である．CafeOBJ は強く型付けされた言語である．

ソートは [...] を使って宣言する．

[⟨*sort-name*⟩...⟨*sort-name*⟩]

⟨*sort-name*⟩ にはソート名を記述する．ソート名をスペースで区切って，複数のソートを一度に宣言することができる．ソート名は、Nat のように頭文字を大文字で記述することが多い．

ソート間の包含関係

CafeOBJ では，包含関係を持つソートを宣言することができる．例えば，自然数と整数を表すソート Nat と Int がある．CafeOBJ では，Nat は Int に含まれる，と宣言することができる．あるソートに含まれるソートをサブソート (subsort)，あるソートを含むソートをスーパーソート (supersort) と呼ぶ．先程の例では，Nat は Int のサブソートであり，Int は Nat のスーパーソートである．なお，ソート間の包含関係は循環してはならない．

[⟨*subsort-name*⟩ < ⟨*supersort-name*⟩]

輸入

あるモジュールで，別の定義済みモジュールの宣言を使用できるようにすることを，モジュールの輸入 (import) と呼ぶ．

モジュールの輸入には，protecting，extending，using(およびそれぞれの省略形pr，ex，us) を用いる：

$pr(\langle module\text{-}exp \rangle)$
$ex(\langle module\text{-}exp \rangle)$
$us(\langle module\text{-}exp \rangle)$

$\langle module\text{-}exp \rangle$ はモジュール式と呼ばれる式であり，輸入するモジュール名を指定する他，様々な指定が行える．

- pr :モデルに新しい要素を付け加えたり，異なっていた 2 つの要素を等しいものとしない．
- ex :モデルに新しい要素を付け加えることを許すが，異なっていた 2 つの要素を等しいものとしない．
- us :特に制限はない．

演算子

演算子は op を使って宣言する．

$op\ \langle op\text{-}name \rangle : \langle arity \rangle\ -> \langle sort\text{-}of\text{-}op \rangle \{\langle operator\text{-}attribute \rangle\}$

$\langle op\text{-}name \rangle$ には演算子名，$\langle arity \rangle$ にはアリティ(arity)，$\langle sort\text{-}of\text{-}op \rangle$ には演算子のソート (sort of operator) を指定する．アリティとは引数に対応する概念であり，ソート名の列である．演算子のソートとは返戻値の型に対応する概念である．アリティと演算子のソートの組をランク (rank) と呼ぶ．

同じランクを持つ複数の演算子は ops を使って一度に宣言できる．

$ops\ \langle op\text{-}name \rangle \ldots \langle op\text{-}name \rangle : \langle arity \rangle\ -> \langle sort\text{-}of\text{-}op \rangle \{\langle operator\text{-}attribute \rangle\}$

演算子の属性

演算子には，以下の属性を指定できる：

- assoc :結合律
- comm :交換律
- r-assoc :右結合
- l-assoc :左結合
- idem :べき等律

等式

指標 (Signature; ソート宣言と演算子宣言の組) から作られる項の集合を T とする．等式は 2 つの項 $t1,\ t2\ \in T$ 間の等価関係を宣言する．仕様中に記述された等式は，公理として推論の際に使用される．

等式は eq を使って宣言する．

$$eq\ \langle lhs \rangle\ =\ \langle rhs \rangle\ .$$

$\langle\ lhs\ \rangle$ と $\langle\ rhs\ \rangle$ は同じソート上の項である．これらの項には変数を含めることができる．変数は特定のソート上に宣言され，そのソート上の任意の項を代入できる．

例として，自然数の仕様に現れる次の等式を例に考える．

$$eq\ 0\ +\ M : Nat\ =\ M\ .$$

M は Nat 上の変数である．M には Nat 上の任意の項を代入できるので，$0+0$ という項は，M に 0 を代入すると，この等式を用いて 0 と等しいと推論できる．

条件付き等式

条件付き等式は ceq を使って宣言する．

$$ceq\ \langle lhs \rangle\ =\ \langle rhs \rangle\ if\ \langle condition \rangle\ .$$

$\langle condition \rangle$ は内臓モジュール BOOL で宣言されているソート Bool 上の項である．

変数

前述したように，変数は等式中で宣言することができる．しかし，頻繁に使う変数はあらかじめまとめて宣言しておくと便利である．このように宣言した変数のスコープは，宣言したモジュール内である．

$$var\ \langle \text{var-}name \rangle\ :\ \langle \text{sort-}name \rangle$$

同じソート上の複数の変数は vars を使って宣言できる：

$$vars\ \langle \text{var-}name \rangle\ ...\ \langle \text{var-}name \rangle\ :\ \langle \text{sort-}name \rangle$$

2.2.2 記述方法

CafeOBJ での仕様の記述方法について具体例を用いて解説する．以下に自然数の仕様を記述した CafeOBJ の仕様の例を示す．

```
mod! SIMPLE-NAT {
  [ Nat ]
  op 0 : -> Nat
  op s : Nat -> Nat
}

mod! NAT+ {
  pr(SIMPLE-NAT)
  op _+_ : Nat Nat -> Nat
  eq 0 + M:Nat = M .
  eq s (N:Nat) + M:Nat = s (N + M) .
}
```

”mod! SIMPLE-NAT”により，SIMPLE-NAT というモジュールの定義が開始する．”[Nat]”によりソートの宣言を行う．Nat は自然数全体を表すソートで，要素として 0 以上の任意の自然数を持つ．ソートの宣言の次に演算を定義している．0 はアリティに何も持たないため定数となることが分かる．”op s”はアリティに Nat を 1 つとり，Nat を返す演算 s を定義している．この演算は，自然数を 1 つ受け取り，それに 1 を足した自然数を返すことを意味している．

さらに，”mod! NAT+”により，NAT+というモジュールの定義を開始している．”pr(SIMPLE-NAT)”によって，先程定義した SIMPLE-NAT モジュールの輸入を行っている．”op _+_”はアリティに 2 つの自然数をとり，1 つの自然数を返す自然数上の加算演算＋を定義している．

次は等式の定義を行っている．シグネチャ部だけでは”0”と”0 + 0”は異なるものとして区別されている．これらが等しいものであるということは等式によって定義される．”eq 0 + M:Nat = M”によって等式で使用する変数 M(ソートは Nat) を宣言し，0 に任意の自然数 M を足したものは M に等しいということを定義している．次は，1 以上の任意の自然数 N と 0 以上の自然数 M に対する等式を定義している．

2.2.3 仕様の検証

CafeOBJ 処理系では，宣言された等式を左辺から右辺への書き換え規則とみなし，与えられた項を書き換え規則を使って次々に書き換え，簡約を行う．それにより仕様の検証を行うことができる．CafeOBJ で記述した仕様がある性質を満たすことを証明したい場合には，証明譜 (proof score) を作成する．証明譜とは，ある仕様において，ある表明が成立することを CafeOBJ に証明させるために指示を記述したテキストである．証明譜には，主に次のようなことを記述する：

- 証明の使用するモジュールの指定

- あるソート上の任意の要素を表す定数 (=変数のように振舞う)
- 場合分け
- 補題
- (帰納法を用いるならば) 帰納法の仮定
- 簡約の指示

ここでは例として自然数の仕様において，+の結合律が成立することを証明する証明譜を示す．結合律とは任意のi，j，kについて，$(i+j)+k=i+(j+k)$が成立することである．

```
open NAT+ + EQL
-- declaring three arbitrary numbers i,j and k on the sort Nat:
ops i j k : -> Nat .

-- base step:
red ((0 + j) + k) = (0 + (j + k)) .
--> should be true.

-- induction hypothesis:
eq (i + j) + k = i + (j + k) .
-- inductive step:
red ((s(i) + j) + k) = (s(i) + (j + k)) .
--> should be true.
close
```

証明を行う際にはopenコマンドを用いる．openコマンドを使用すると，証明に用いる定数や等式をモジュールに追加できるようになる．次に定数を宣言し，”red (0 + j) + k = 0 + (j + k)”により，基底 $(i=0)$ の推論を行う．そして帰納段階の推論を行う．つまり，任意のiについて$(i+j)+k=i+(j+k)$が成立すると仮定したとき，iが1増えたとき (すなわちs(i)) も表明が保存されることを示す．”close”によってオープンしたモジュールをクローズする．クローズすると，追加した定数や等式は破棄される．

この証明譜をCafeOBJ処理系に読み込ませ，実行した結果を以下に示す．

```
-- opening module NAT+ + EQL.. done._*
-- reduce in %NAT+ + EQL : (0 + j) + k = 0 + (j + k)
true : Bool
(0.003 sec for parse, 3 rewrites(0.002 sec), 11 matches)
```

```
--> should be true.*
-- reduce in %NAT+ + EQL : (s(i) + j) + k = s(i) + (j + k)
true : Bool
(0.004 sec for parse, 5 rewrites(0.003 sec), 26 matches)
--> should be true.
```

基底と帰納段階の推論を行う 2 回の簡約のいずれも期待した結果 (true) が得られたので，+には結合律が成立することが証明できたことがわかる．CafeOBJ 処理系は，定義した仕様を高効率で簡約することが可能であるという特徴を持っており，大規模な仕様に対しても十分な処理能力を持っている．

第3章　短距離無線通信規格NFC

この章では，仕様化の対象とした短距離無線通信規格 NFC について述べる．

3.1　NFC概要

NFC(Near Field Communication)[7] とは，ソニーと NXP(旧フィリップス) が共同で開発した，短距離無線通信規格であり，2003 年 12 月に ISO/IEC IS 18092 として国際基準となった．非接触型識別技術と相互接続技術を組み合わせた無線通信規格で，13.56MHz の周波数を使用し，約 10cm 以内という，近距離で 106kbps〜424kbps の速度での通信が可能となる．NFC はソニーが開発した FeliCa や，欧米で広く利用されている Mifare と通信互換性があり，FeliCa や Mifare が単方向通信であるのに対して，NFC は双方向通信が可能である．NFC の通信モードは受動モード，能動モードに分けられる．

- 受動モード

  FeliCa や Mifare と通信互換性のあるモードで，通信を行いたい機器 (Initiator) が自ら無線通信フィールドを展開して通信を始める．通信対象 (Target) は Initiator の展開した無線フィールドを用いて要求に応える．

- 能動モード

  Initiator と Target のどちらも自らの無線フィールドを用いて通信を行う．Initiator は自らの無線フィールドを用いて要求 (REQ) を送った後，自分の無線フィールドを切断し，その後 Target は自分の周りに他の無線フィールドがないことを確認し，自らの無線フィールドを展開し，要求に対する返答を行う．

3.2　NFC仕様

NFC プロトコルの流れを述べる．初期設定として，NFC に基づく端末は Target mode である．Target mode であれば，自ら電波を発することはなく，発信者 (Initiator) からのコマンドを静かに待つ．アプリケーションの要求により NFC 端末が Initiator mode に切り替わり，受動モードか能動モードが選択され，転送速度も決定される．Initiator は自分の周りの電波の存在を確かめ，周りに他の無線フィールドがあれば，無線フィールドを起動しない．もし，周りに無線フィールドがなければ，Initiator は無線フィールドを起

動させる．Initiator の無線フィールドにより，通信対象 (Target) は起動される．そして，Initiator から Target へコマンドが送信され，Target から Initiator へコマンドに対する返答がなされる．

3.2.1 初期設定

この節では，通信を行う前の初期設定と衝突回避プロトコルについて述べる．少なくとも二つの目標者が同時にデータを転送した時に，発信者は衝突を知ることができる．

衝突回避は他の NFC の通信を妨げないようにするために行われる．NFC 通信の発信者は，他の通信が行われているのであれば，自身の通信を開始しない．

Target と通信を始めるために，Initiator は周りの無線フィールドを連続的に監視している．もし，$T_{IDT} + n \times T_{RFW}$ の間，周りに無線フィールドがなければ，自身が通信を始める．n はランダムな整数である．図 3.1 に初期衝突回避を示す．

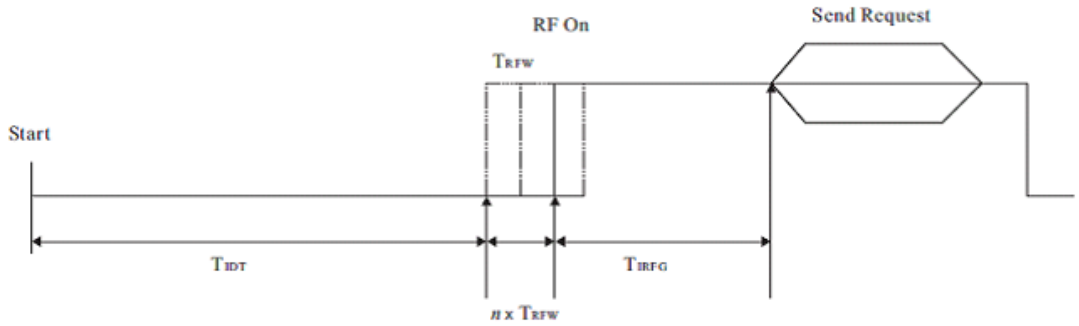

図 3.1: 初期衝突回避

T_{IDT} ： 初期遅延時間 $T_{IDT} > \frac{4096}{f_c}$
T_{RFW} ： 無線待ち時間 $\frac{512}{f_c}$
n ： T_{RFW} の時間間隔の乱数 $0 \leq n \leq 3$
T_{IRFG} ： 無線フィールドオンとデータを送り始める間の時間 $T_{IRFG} > 5ms$

乱数 n によって，他の Initiator と電波の衝突が起こらないようにタイミングをずらしている．能動モードであれば，Initiator によって作られた無線フィールドはオフになり，受動モードであれば，Initiator によって作られた無線フィールドはオンのままである．

能動モード時には，一つ以上の Target から Initiator への応答衝突回避が行われる．図 3.2 に初期設定時の応答衝突回避を示す．

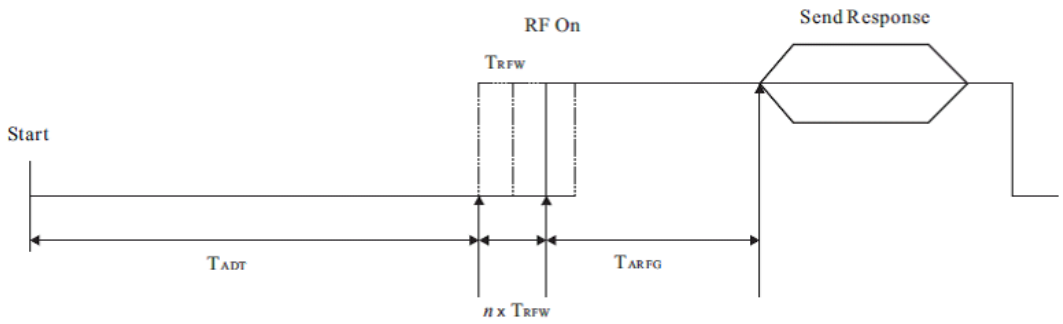

図 3.2: 応答衝突回避

T_{ADT} ： 遅延時間 (Initiator と Target の電波を感知する時間) $\frac{768}{f_c} \leq T_{ADT} \leq \frac{2559}{f_c}$
T_{RFW} ： 無線待ち時間 $\frac{512}{f_c}$
n ： T_{RFW} の時間間隔の乱数 $0 \leq n \leq 3$
T_{ARFG} ： 無線フィールドオンとデータを送り始める間の時間 $T_{ARFG} > \frac{1024}{f_c}$

以下，受動モード，能動モードそれぞれの初期設定を述べる．

受動モード

受動モードで扱われる Target の状態について述べる．

- SENSE State

  この状態は, Initiatorからのコマンド待ち状態であり, SENS_REQ もしくは ALL_REQ コマンドを認識する．もし，有効な SENS_REQ か ALL_REQ を受け取ったなら，SENS_RES を送信し，RESOLUTION State へ移行する．SENSE State 時にその他のコマンドを受け取っても，SENSE State のままである．

- RESOLUTION State

  この State で SDD が適応される．また，Cascade levels も扱われる．もし，完全な NFCID を含む有効な SEL_REQ を受け取ったなら，Initiator に SEL_RES を送信し，SELECTED State へ移行する．RESOLUTION State 時にその他のコマンドを受け取った場合は SENSE State へ移行する．

- SELECTED State

  この State では，Target は ATR_REQ もしくは有効な命令コマンドを待っている．もし，有効な SLP_REQ を受け取ったなら，SLEEP State へ移行する．トランスポートプロトコルでは，DSL コマンドが Target を SLEEP State へ移行させるコマンド

である．SELECTED State時にその他のコマンドを受け取った場合はSENSE State へ移行する．

- SLEEP State

  この状態では，Targetは ALL_REQ コマンドのみに反応し，RESOLUTION* State へ移行する．Target は有効な ALL_REQ を受け取ったら，SENS_RES を送信して RESOLUTION* State へ移行する．SLEEP State時にその他のコマンドを受け取っても，SLEEP State のままである．

- RESOLUTION* State

  この RESOLUTION* State は RESOLUTION State と似ており，SDD が適応され，Cascade levels も扱われる．Target は自身の完全な NFCID が選択された時に SELECTED* State へ移行する．RESOLUTION* State時にその他のコマンドを受け取った場合はSLEEP State へ移行する．

- SELECTED* State

  この SELECTED* State は SELECTED State と似ている．すなわち，Target は ATR_REQ もしくは有効な命令コマンドを待っており，もし，有効な SLP_REQ を受け取ったなら，SLEEP State へ移行する．トランスポートプロトコルでは，DSL コマンドが Target を SLEEP State へ移行させるコマンドである．SELECTED* State 時にその他のコマンドを受け取った場合はSLEEP State へ移行する．

受動モードでやりとりされるメッセージについて述べる．

- SENS_REQ と ALL_REQ コマンド

  SENS_REQ と ALL_REQ コマンドは無線フィールドに Target が存在するかを調べるために，Initiator によって送られる．ALL_REQ はInitiator によって送られ，Target を SLEEP State から RESOLUTION* State へ移行させるために使われる．そして，SDD を行っていく．

- SENS_RES

  Initiator によって SENS_REQ コマンドが送られた後，SENSE State の全ての Target は SENS_RES を送って反応する．Initiator によって ALL_REQ コマンドが送られた後，SENSE もしくは SLEEP State の全ての Target は SENS_RES を送って反応する．Initiator は複数の Target が反応した時，衝突が起こったことがわかる．

- SDD_REQ と SEL_REQ コマンド

  これらのコマンドはSDD を行っている間に使用されるコマンドで，SEL_CMD，SEL_PAR，そして SEL_PAR の値に従った 0 から 40 ビットのデータから成る．SEL_CMD は

Cascade levels を示す CLn を表している．SDD_REQ と SEL_REQ の違いは，データビットが 40 ビットか否かの違いであり，有効な 40 ビットであれば，そのコマンドは SEL_REQ である．このコマンドが SDD_REQ である限り，Target は RESOLUTION または RESOLUTION* State を維持する．もし，Target が完全な NFCID を受け取ったら，SELECTED または SELECTED* State へ移行して，SEL_RES を返信する．そうでなければ，Target は RESOLUTION または RESOLUTION* State を維持し，Initiator は CLn を増やして，新たに SDD を行う．

以下に初期設定と SDD のフローチャートを示す.

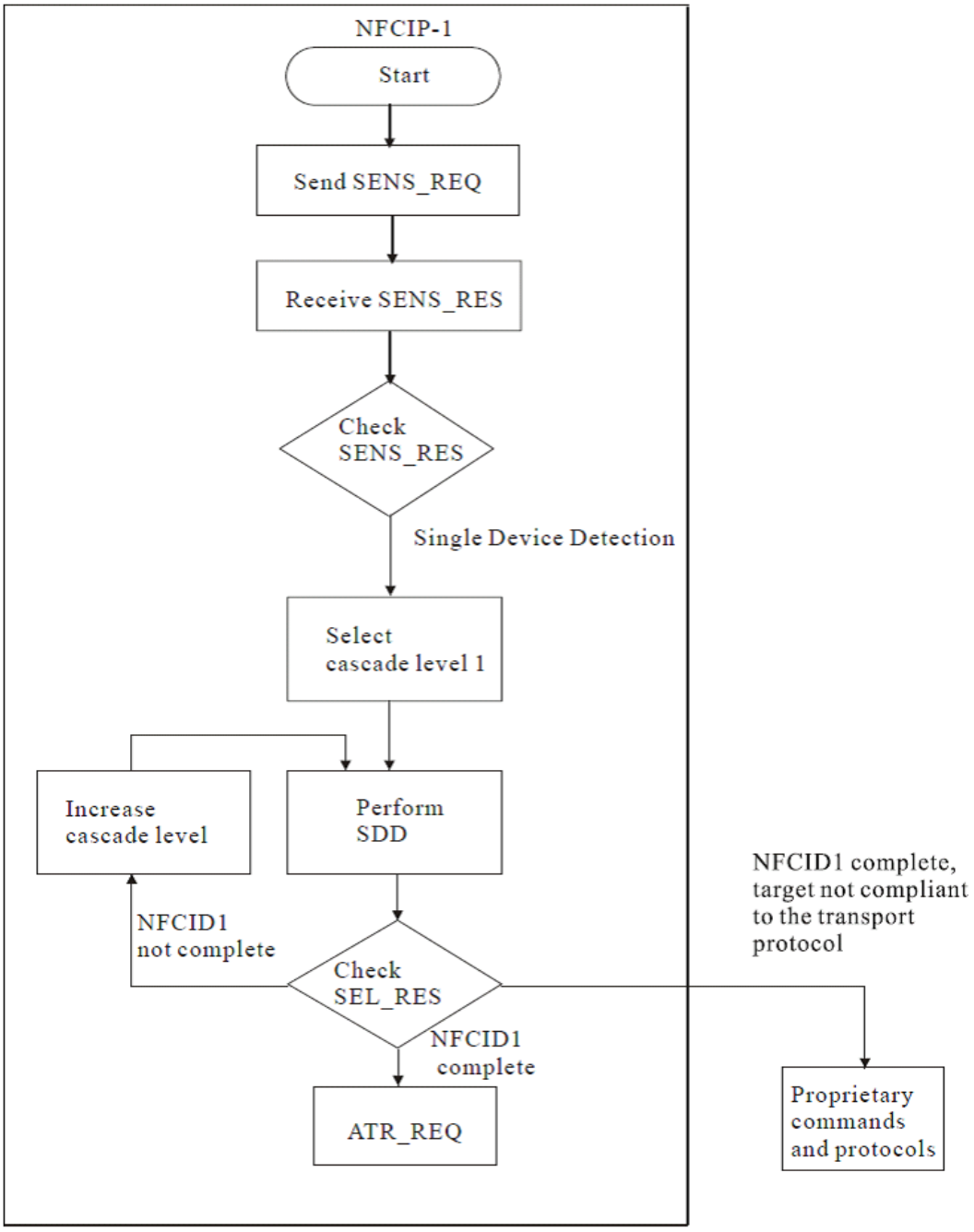

図 3.3: フローチャート：初期設定と SDD

能動モード

能動モードでの衝突回避は図 3.4 のように行われる．

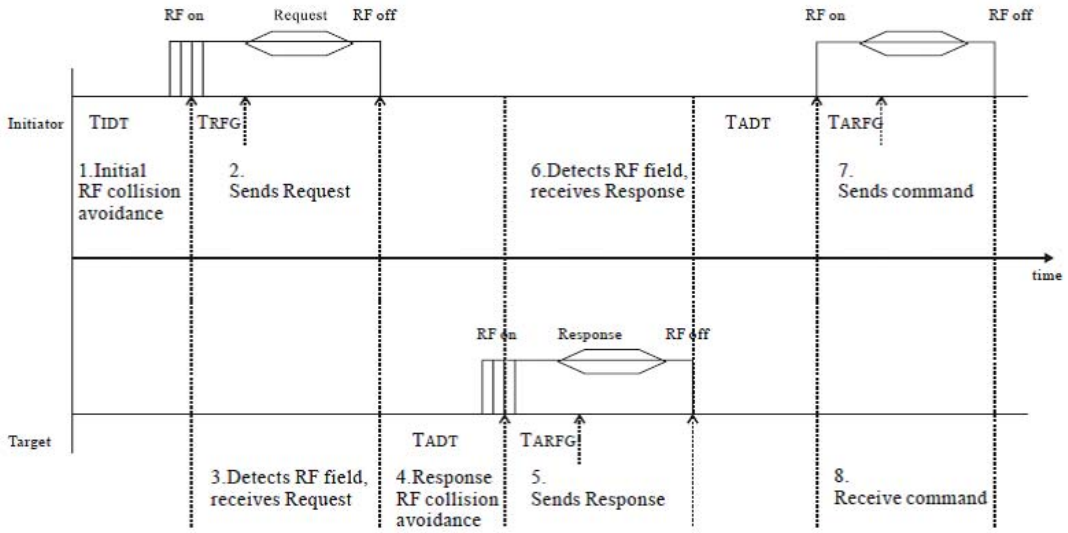

図 3.4: 能動モード時の衝突回避

Initiator は初期衝突回避を行う．その後，Initiator によって最初のコマンドである ATR_REQ が送られる．コマンド送信後，Initiator は無線フィールドをオフにする．そして，Target は応答衝突回避を行い，ATR_REQ の応答として ATR_RES を送り，無線フィールドをオフにする．Initiator はタイムスロットを 0 して，パラメータ変更のための PSL_REQ，もしくは，データ交換プロトコルを始めるために DEP_REQ を送る．フィールドに 2 つ以上の Target がいる場合は，タイムスロットが最も低い Target が最初に反応し，他の Target は反応しない．もし，2 つ以上の Target が同じタイムスロットで同時に通信を行ってきた場合は，Initiator はデータが衝突したことがわかるので，もう一度 ATR_REQ を送り，反応を待つ．

3.2.2 SDD

SDD(Single Device Detection) は，受動モード時に，Initiator が複数の Target から 1 つの Target を通信相手に選択するためのアルゴリズムである．SDD にはそれぞれの Target がランダムに生成した NFCID，Cascade level が扱われる．Cascade level は 1 から 3 である．NFCID はそれぞれ 4，7 もしくは 10 バイトである．以下に SDD の処理ステップを示す．

1. Initiator は SEL_CMD に Cascade level のコードを割り当てる．

2. Initiator は SEL_PAR に 16 進数の 20 を割り当てる．この値は Initiator が NFCID を獲得していないことを表している．このコマンドはフィールドにいる Target 全てが自身の NFCID を返信するようにさせる命令である．

3. Initiator は SEL_CMD と SEL_PAR を送信する．

4. フィールドにいる全ての Target は自身の NFCID を Initiator に送信する．

5. フィールド内にいる Target は異なった NFCID を持っていると予想され，2 つ以上の Target が反応すれば衝突が起こる．衝突が起こらなければステップ 6 から 10 はとばす．

6. Initiator は最初の衝突が起こったところを認識する．

7. Initiator は NFCID の有効ビットの数に応じた値を SEL_PAR に割り当てる．NFCID の有効ビットは，衝突が起こる前までに獲得した NFCID に 0 か 1 を加えたものである．

8. Initiator は SEL_CMD と SEL_PAR に続けて，獲得した有効ビットを送信する．

9. Initiator によって送られた有効ビットと等しい NFCID の一部分をもった Target のみが自身の NFCID を送信する．

10. もし，衝突が起こったら，ステップ 6 から 9 を繰り返す．ループの最大は 32 とする．

11. もし，衝突か起こらなければ，Initiator は SEL_PAR に 16 進数の 70 を割り当てる．この値は Initiator が完全な NFCID を送信することを表している．

12. Initiator は SEL_CMD と SEL_PAR に続けて，40 ビットの NFCID，そしてエラー検出のための周期冗長検査コード CRC を送信する．

13. NFCID が Initiator の送った 40 ビットと一致した Target は SEL_RES で反応する．

14. もし NFCID が完全であれば，Target は Cascade bit をつけた SEL_RES を送り，RESOLUTION State から SELECTED State，または RESOLUTION* State から SELECTED* State へ移行する．

15. Initiator はもう一度 SDD を行うかを決めるために，SEL_RES 内の Cascade bit を確認する．Cascade bit が 1 であれば Cascade level をあげて SDD を行う．

このアルゴリズムは完全な NFCID を獲得するために Initiator によって用いられる．上記のように SDD は行われ，複数の Target から一つの Target を選択する．

3.2.3 プロトコルの有効化

ここではデータ交換プロトコルを行う前に行われる Target の属性のための要求や，パラメータ変更について述べる．

受動モード

受動モードのプロトコル有効化について述べる．

1. Initiator は初期衝突回避を行う．
2. Initiator は選ばれた転送速度で受動モードの初期設定と SDD を行う．
3. 属性要求に従って，NFCIP-1 プロトコルのサポートを確認する．
4. ATR_REQ がサポートされていなければ，Target は初期設定と SDD に戻る．
5. Initiator は Target が属性要求の利用可能であると受け取った後，次のコマンドとして ATR_REQ を送る．
6. Target は選択された後に，直接 ATR_REQ を受け取った時のみ ATR_RES を返す．
7. もし，Target がパラメータ変更をサポートしていれば，次のコマンドとして Initiator によって PSL_REQ が送られる．
8. Target は PSL_REQ の返信として PSL_RES を送る．
9. ATR_RES 内でパラメータ変更をサポートしていないなら，Target はパラメータ選択を必要としない．
10. 透過データはデータ交換プロトコルで送られる．

受動モードの Initiator の有効化の流れを図 3.5 に示す．

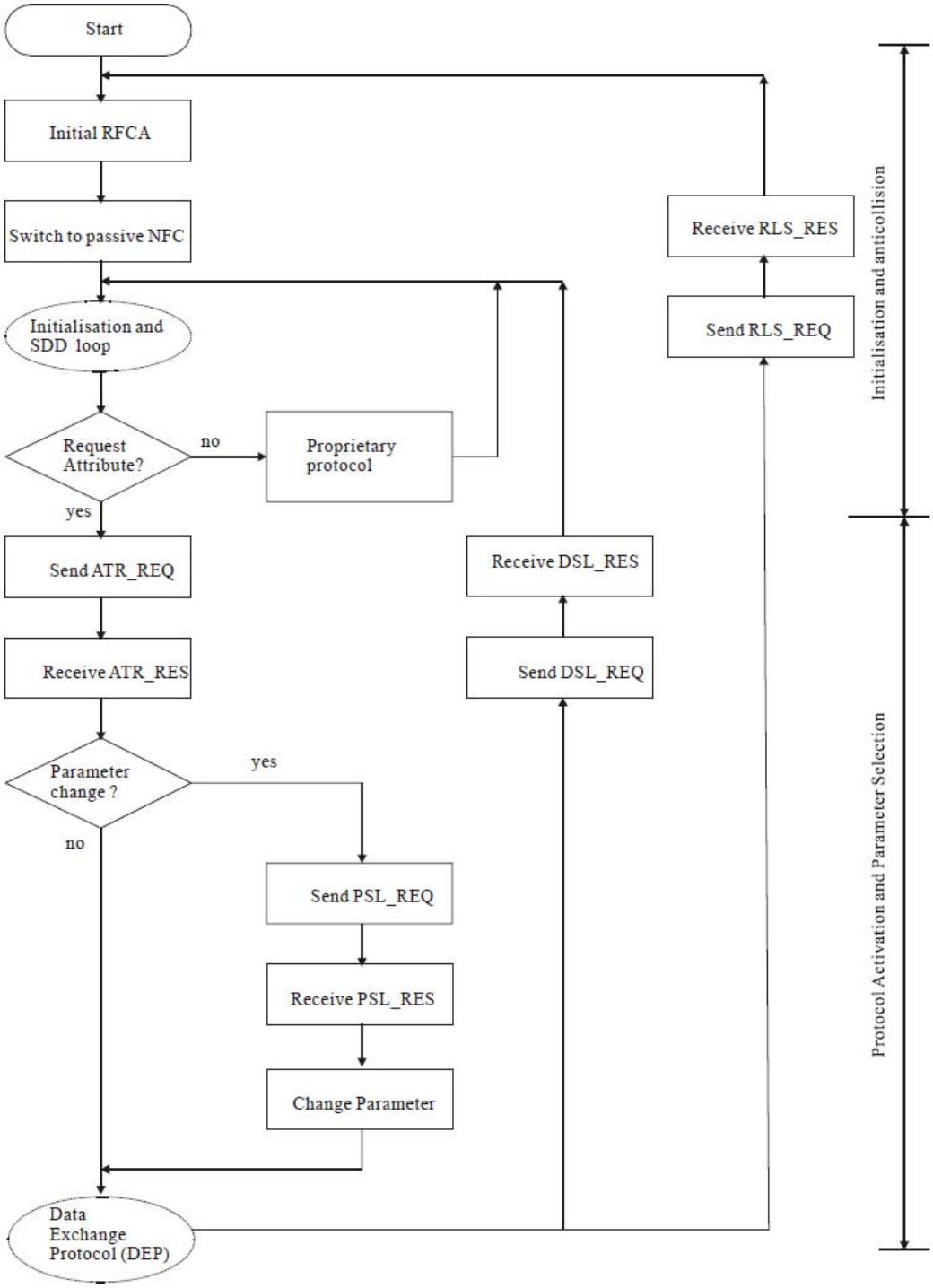

図 3.5: 受動モード時の有効化プロトコル

能動モード

能動モードのプロトコル有効化について述べる．

1. Initiator は初期衝突回避を行う．
2. Initiator は能動モードで選択された転送速度に切り替える．
3. Initiator は ATR_REQ を送信する．
4. Target は ATR_REQ の返信として ATR_RES を送る．この ATR_RES が成功すると Target は選択される．
5. もし，Initiator がデータの衝突を発見したならば，ATR_REQ を再送する．
6. もし，Target がパラメータ変更をサポートしていれば，Initiator は ATR_RES を受け取った後，パラメータ変更のために PSL_REQ を送る．
7. Target は PSL_REQ の返信として PSL_RES を送る．
8. ATR_RES 内でパラメータ変更をサポートしていないなら，Target はパラメータ選択を必要としない．

受動モードの Initiator の有効化の流れを図 3.6 に示す．

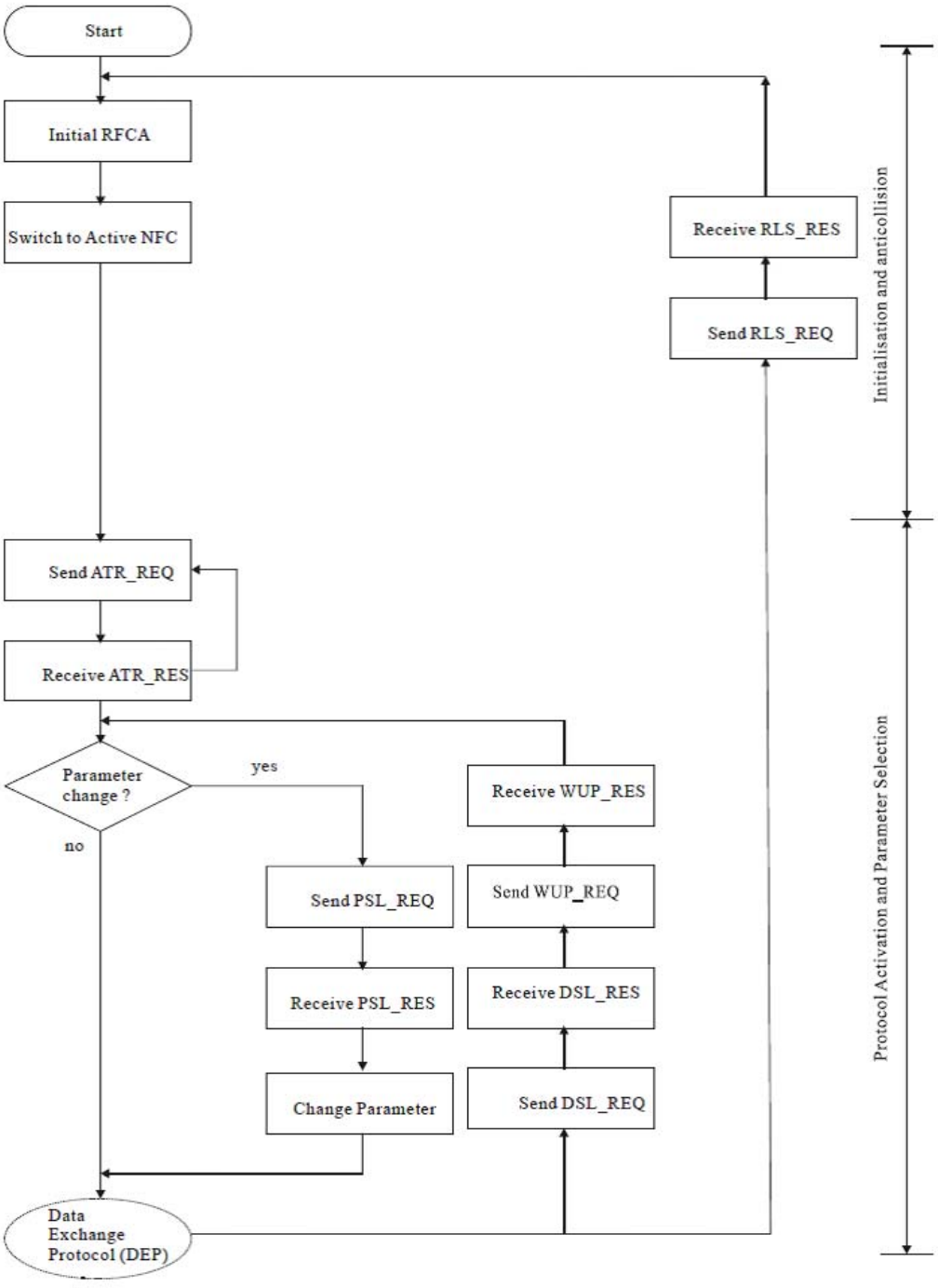

図 3.6: 受動モード時の有効化プロトコル

有効化時に扱われるメッセージについて述べる．

- ATR_REQ と ATR_RES

  ATR_REQ と ATR_RES は NFCID や送受信ビットレート等を含んだデータである．Initiator は ATR_REQ を送信し，有効な ATR_RES を受け取った時，処理を続ける．Target が有効な ATR_REQ を受け取ったら，ATR_RES を送信し，次の ATR_REQ を無効にする．Target が他のフレームや無効なフレームがきたら無視し，受信状態を維持する．

- WUP_REQ と WUP_RES

  WUP_REQ と WUP_RES は能動モードのみに適応されるデータで異なる Target を再起動するために使われる．Initiator は WUP_REQ を送信し，有効な WUP_RES を受け取った時，処理を続ける．最後のコマンドにより選択を解除された Target が，その NFCID を持った WUP_REQ を受け取ったならば，WUP_RES を送信し，次の WUP_REQ を無効にする．もし，その他のフレームを受け取ったなら無視し、受信状態を維持する．

- PSL_REQ と PSL_RES

  PSL_REQ と PSL_RES はビットレートやフレームの最大長等を含んだデータで，パラメータ変更を行うために使われる．Initiator は有効な PSL_RES を受信したら，定義されているフォーマットを変更し，処理を続ける．Target が有効な PSL_REQ を受け取ったら，PSL_RES を送信し，次の PSL_REQ を無効にする．また，定義された値にパラメータを変更し，受信状態を維持する．無効なフレームを受け取ったら，無視し，PSL_REQ を無効にする．そして，現在のフレーミングのまま受信状態を維持する．もし，PSL_REQ 以外の有効なフレームを受け取ったら，PSL_REQ を無効にし，現在のフレーミングで処理を続ける．

3.2.4 データ交換プロトコル

データ交換プロトコルは，エラー処理を含んだ半二重プロトコルである．ヘッダー部分に続いてデータが送られる．ヘッダー部分は，DEP_REQ か DEP_RES を現すコマンド，転送をコントロールする PFB，デバイス ID 等を含む．転送をコントロールする PFB は 3 つのタイプに分けられる．

- Information pdu

  アプリケーション層への情報を表している PFB で，PNI(Packet Number Information) を含む．PNI は Initiator から Target へ送られたパケットの数をカウントするものである．

PNI は 0 から始まり，Initiator が保持している PNI と等しい Information もしくは ACK pdu を受け取った場合は，Initiator は新しいフレームを送信する前に，現在の PNI を増やす．Target が保持している PNI と等しい Information もしくは ACK pdu を受け取った場合は，この PNI で応答を行い，その後 PNI を増やす．

- ACK/NACK pdu

  肯定か否定かを現す PFB．PNI も含む．

- supervisory pdu

  これは Target の存在を確認したり，タイムアウトを現す PFB．

最初の pdu は Initiator によって送られる．Information pdu を含んだデータを受け取った場合は，ACK pdu を含むデータで処理される．

Initiator は，仕様と一致しない pdu を受け取った場合は NACK pdu を送信する．もしタイムアウトが起こったなら，Target の存在を確認する attention コマンドが送られる．タイムアウトが起こって，前に NACK を送っているならば，NACK を再送する．PNI が等しい ACK pdu を受け取ったなら，処理を続ける．

一方，Target は，仕様と一致しない pdu を受け取った場合はなにもせず，状態も変化しない．もし，NACK pdu を受け取った時，その PNI が前に送った pdu の PNI と等しい場合には，前のブロックを再送する．attention コマンドには，attention res を返す．

3.2.5 プロトコルの終了

データ交換プロトコルを用いたデータの交換の後，Initiator はデータ交換プロトコルを終了させる．プロトコルの終了が成功した場合は Initiator と Target は初期状態に戻る．また，Target の再起動が可能であるが，その再起動は受動モードと能動モードで異なる．受動モードでの再起動は Initiator は Target に ALL_REQ を送信し，能動モードでは WUP_REQ を送信する．

プロトコルの終了には DSL コマンドと RLS コマンドを用いる．

- DSL_REQ と DSL_RES

  Initiator が DSL_REQ を送信し，有効な DSL_RES を受け取った時，Target の選択開放が成功する．Initiator は他の Target と通信を行うことが可能．Target が DSL_REQ を受け取って，DSL_RES を送信すれば，Target は初期状態へ移行する．

- RLS_REQ と RLS_RES

  Initiator が RLS_REQ を送信し，有効な RLS_RES を受け取った時，Target の選択開放が成功する．Target が RLS_REQ を受け取って，RLS_RES を送信すれば，Initiator と Target は共に，初期状態へ移行する．

このようにデータ交換プロトコルの終了が行われる．

第4章　形式手法によるNFCの仕様記述

本研究では代数仕様記述言語 CafeOBJ を用いて，OTS/CafeOBJ 法により仕様化・検証を行う．CafeOBJ には，可視ソートと隠蔽ソートの 2 種類のソートがあり，可視ソートは抽象データ型を，隠蔽ソートは抽象機械の状態空間を現すのに用いる．さらに，隠蔽ソートに関連して 2 種類の演算，観測演算と遷移演算がある．観測演算は抽象機械の状態を観測するのに用いられ，抽象機械の状態と 0 個以上のデータを引数にとり，その状態に関連する値を返す．遷移演算は抽象機械の状態を変化させるのに用いられ，抽象機械の状態と 0 個以上のデータを引数にとり，抽象機械の変化後の状態を返す．状態遷移は，抽象機械の状態に遷移演算を適用する前後で各観測演算の返戻値がどのように変化するのかを，等式を用いて定義される．OTS/CafeOBJ 法はある状態における値を観測するための観測関数，状態を更新させるための遷移関数を用いた仕様記述及び検証の方法である．観測関数，遷移関数を用いてシステムの振舞をモデル化したものを観測遷移機械 OTS(Observation Transition System) と呼ぶ．OTS は $\langle O,I,T\rangle$ により定義され，それぞれ O は観測関数の集合，I は初期状態の集合，T は遷移規則の集合を現す．OTS/CafeOBJ 法を用いて，NFC のデータ転送の安全性や信頼性を解析し，検証を行う．

プロトコルの振舞

NFC プロトコルの動作は前述したとおりであるが，初めに Initiator は，他の通信が行われていないことを確認した後，SENSE コマンドにより，Target の存在を確認する．Target の返信により，Target がフィールドに存在しているということを確認した後，フィールド内の Target の NFCID を獲得するために SDD コマンドを送信する．このコマンドはフィールドに存在する Target 全てが返信を行うような命令であるので，複数の Target からの返信が来る場合がある．この Target からの返信は，Target の ID 情報を送信しているものである．複数の Target からの返信を Initiator が確認すれば，SDD アルゴリズムにより，複数の Target から 1 つの Target を選択する．Initiator が Target の NFCID を獲得した場合は，その Target の NFCID を用いて通信を行う．Initiator の送った NFCID が異なる Target は反応せず，コマンドを無視する．Initiator と選択された Target はパラメータ変更などの設定を行い，データ転送を行う．データ転送が終了すると，Initiator は Target を解放するために DSL，もしくは RLS コマンドを用いる．Target は自ら動作することはなく，Initiator の命令に反応する．

本研究では，NFC プロトコルを抽象化して仕様を作成し，検証を行った．具体的には，SENSE コマンドによって Target の存在を確認し，SDD コマンドによって，Target

の NFCID を獲得し，獲得した NFCID により Target の選択を行う．選択した Target には SEL コマンドを送信し，Target は SEL コマンドの返信を行う．そしてその後，Target を解放するために DSL コマンドを用いる．すなわち，Target 選択後のパラメータ変更等，データ転送は行わず，NFC プロトコルの全体的な流れを形式化した．今回作成した仕様の解説は以下で行う．

Initiator が 1 に対して，複数の Target が存在するとするとし，送った Msg は途中でなくならず，必ず 1 つ以上の Target に届くことを想定している．Initiator は sendermsg，Target は imsg をそれぞれ持ち，受け取った Msg を格納する．また，Initiator は sendergetid を持ち，Target の ID を格納する．Initiator はまず sen_req を送り，それを受け取った Target は sen_res を返す．Initiator は sen_res が返ってくれば，相手の ID 獲得の為に Initiator は sdd_req を送信する．sdd_req を受け取った Target は自身の ID と共に sdd_res を送る．sdd_res を受け取った Initiator はその ID 宛てに sel_req を送信する．sel_req を受け取った Target は，sel_req 内の ID が自身の ID と等しいことを確認し，sel_res を返す．その後，Initiator は Target を解放するために選択した ID 宛てに dsl_req を送信する．dsl_req を受け取った Target は，dsl_req 内の ID が自身の ID と等しいことを確認し，dsl_res を返す．dsl_res を送信した Target は初期状態へ移行する．

以下に作成したデータ型の CafeOBJ 仕様を示す．

個体識別の仕様 Id：個体識別の型 Id のモジュールは以下のとおりである．

```
mod* ID{
  [Id]
  ops sender : -> Id
  op _=_ : Id Id -> Bool {comm}
  vars I J : Id
  eq (I = I) = true .
  eq (sender = I) = false .
}
```

定数 sender は Initiator を示す．

メッセージの仕様 Msg：メッセージの型 Msg のモジュールは以下のとおりである．

```
mod* MSG{
  [Msg]
  ops sen sdd sel dsl none : -> Msg
  op _=_ : Msg Msg -> Bool {comm}
  var M : Msg
  eq (M = M) = true .
```

```
  eq (sen = sdd) = false .
  eq (sen = sel) = false .
  eq (sen = dsl) = false .
  eq (sen = none) = false .
  eq (sdd = sel) = false .
  eq (sdd = dsl) = false .
  eq (sdd = none) = false .
  eq (sel = dsl) = false .
  eq (sel = none) = false .
  eq (dsl = none) = false .
}
```

MSG モジュールでは，やり取りされるメッセージを宣言している．sen は初期設定時に扱われる SENS メッセージを現す．sdd は SDD 時に扱われるメッセージを現す．sel は SDD 後に Initiator が選択した Target へ送信する SEL メッセージを現す．dsl は Initiator が選択した Target を解放するために使われる DSL メッセージを現す．none は Target が初期状態ではメッセージを受け取っていないということを示すための定数である．

NFC の CafeOBJ 仕様 PROTOCOL：PROTOCOL のモジュールは以下のとおりである．

```
mod* PROTO{
  pr(ID)
  pr(MSG)
  *[Sys]*

-- any initial state

  op init : -> Sys

-- observation

  bop senderflag : Sys -> Bool
  bop sendergetid : Sys -> Id
  bop sendermsg : Sys -> Msg
  bop iflag : Sys Id -> Bool
  bop imsg : Sys Id -> Msg

-- actions
```

```
  bop send-sen : Sys -> Sys
  bop rece-sen : Sys Id -> Sys
  bop send-sdd : Sys -> Sys
  bop rece-sdd : Sys Id -> Sys
  bop send-sel : Sys Id -> Sys
  bop rece-sel : Sys Id -> Sys
  bop send-dsl : Sys Id -> Sys
  bop rece-dsl : Sys Id -> Sys

  ...
}
```

定数 init は任意の初期状態を現す．観測関数は sender すなわち Initiator が sel メッセージを送信したかを示す senderflag，sender が獲得した ID を格納する sendergetid，sender が獲得した Msg を格納する sendermsg，ある Target が sender に対して返信可能かどうかを示す iflag，ある Target が獲得した Msg を格納する imsg となっている．

遷移演算は sender(Initiator) から Target への送信は send，Target から sender(Initiator) への送信は rece として記述しており，それぞれ sen，sdd，sel，dsl の 4 種類がある．．．．の箇所に，初期状態および振舞いを定義する等式が宣言される．それらを以降で記述し，それぞれについて解説を行う．

初期状態の定義：任意の初期状態を現す定数 init は以下のとおりに定義される．

```
eq senderflag(init) = false .
eq sendergetid(init) = sender .
eq sendermsg(init) = none .
eq iflag(init,I) = false .
eq imsg(init,I) = none .
```

遷移演算 send-sen の定義：遷移演算 send-sen を定義する等式は以下のとおりである．

```
op c-send-sen : Sys -> Bool
eq c-send-sen(S) = not senderflag(S) .
--
eq senderflag(send-sen(S)) = senderflag(S) .
eq sendergetid(send-sen(S)) = sendergetid(S) .
eq sendermsg(send-sen(S)) = sendermsg(S) .
ceq iflag(send-sen(S),I) = true                 if c-send-sen(S) .
ceq imsg(send-sen(S),I) = sen                   if c-send-sen(S) .
ceq send-sen(S) = S                             if not c-send-sen(S) .
```

send-sen の効力条件は c-send-sen で現されており，senderflag が false の時いつでも可能となる．sender が send-sen を送ることによって，Target は送信可能となり iflag が true となる．また，Target の imsg には sen が格納される．

遷移演算 rece-sen の定義：遷移演算 rece-sen を定義する等式は以下のとおりである．

```
op c-rece-sen : Sys Id -> Bool
eq c-rece-sen(S,I) = iflag(S,I) and imsg(S,I) = sen and not senderflag(S) .
--
eq senderflag(rece-sen(S,I)) = senderflag(S) .
eq sendergetid(rece-sen(S,I)) = sendergetid(S) .
ceq sendermsg(rece-sen(S,I)) = sen
                               if c-rece-sen(S,I) .
ceq iflag(rece-sen(S,I),J) = (if I = J then false else iflag(S,J) fi)
                             if c-rece-sen(S,I) .
eq imsg(rece-sen(S,I),J) = imsg(S,J) .
ceq rece-sen(S,I) = S
                    if not c-rece-sen(S,I) .
```

rece-sen の効力条件は Target が送信可能であり，Target が sen メッセージを格納していることである．これは Target は sender(Initiator) の命令によってのみ動作するためである．

遷移演算 send-sdd の定義：遷移演算 send-sdd を定義する等式は以下のとおりである．

```
op c-send-sdd : Sys -> Bool
eq c-send-sdd(S) = (sendermsg(S) = sen) and not senderflag(S) .
--
eq senderflag(send-sdd(S)) = senderflag(S) .
eq sendergetid(send-sdd(S)) = sendergetid(S) .
eq sendermsg(send-sdd(S)) = sendermsg(S) .
ceq iflag(send-sdd(S),I) = true              if c-send-sdd(S) .
ceq imsg(send-sdd(S),I) = sdd                if c-send-sdd(S) .
ceq send-sdd(S) = S                          if not c-send-sdd(S) .
```

send-sdd の効力条件はフィールドに Target が存在するという sen メッセージを受け取っており，Target を選択していないことである．この send-sdd により，sender は Target から ID を獲得しようとする．

遷移演算 rece-sdd の定義：遷移演算 rece-sdd を定義する等式は以下のとおりである．

```
op c-rece-sdd : Sys Id -> Bool
```

```
eq c-rece-sdd(S,I) = iflag(S,I) and (imsg(S,I) = sdd) and not senderflag(S) .
--
eq senderflag(rece-sdd(S,I)) = senderflag(S) .
ceq sendergetid(rece-sdd(S,I)) = I
                                 if c-rece-sdd(S,I) .
ceq sendermsg(rece-sdd(S,I)) = sdd
                               if c-rece-sdd(S,I) .
ceq iflag(rece-sdd(S,I),J) = (if I = J then false else iflag(S,J) fi)
                             if c-rece-sdd(S,J) .
eq imsg(rece-sdd(S,I),J) = imsg(S,J) .
ceq rece-sdd(S,I) = S
                  if not c-rece-sdd(S,I) .
```

rece-sdd の効力条件は Target が送信可能であり，Target が sdd メッセージを格納していることである．この rece-sdd により，sender は Target の ID を獲得する．

遷移演算 send-sel の定義：遷移演算 send-sel を定義する等式は以下のとおりである．

```
op c-send-sel : Sys Id -> Bool
eq c-send-sel(S,I) = (not senderflag(S)) and (sendergetid(S) = I)
                                         and (sendermsg(S) = sdd) .
--
ceq senderflag(send-sel(S,I)) = true
                                if c-send-sel(S,I) .
eq sendergetid(send-sel(S,I)) = sendergetid(S) .
eq sendermsg(send-sel(S,I)) = sendermsg(S) .
ceq iflag(send-sel(S,I),J) = (if I = J then true else false fi)
                             if c-send-sel(S,I) .
ceq imsg(send-sel(S,I),J) = (if I = J then sel else sen fi)
                            if c-send-sel(S,I) .
ceq send-sel(S,I) = S
                  if not c-send-sel(S,I) .
```

この send-sel により Target を選択する．send-sel を true にする．

遷移演算 rece-sel の定義：遷移演算 rece-sel を定義する等式は以下のとおりである．

```
op c-rece-sel : Sys Id -> Bool
eq c-rece-sel(S,I) = iflag(S,I) and (imsg(S,I) = sel) and sendergetid(S) = I .
--
```

```
eq senderflag(rece-sel(S,I)) = senderflag(S) .
eq sendergetid(rece-sel(S,I)) = sendergetid(S) .
ceq sendermsg(rece-sel(S,I)) = sel
                              if c-rece-sel(S,I) .
ceq iflag(rece-sel(S,I),J) = (if I = J then false else iflag(S,J) fi)
                            if c-rece-sel(S,I) .
eq imsg(rece-sel(S,I),J) = imsg(S,J) .
ceq rece-sel(S,I) = S
                  if not c-rece-sel(S,I) .
```

rece-sel により選択完了となり，sender と Target は通信を行う．

遷移演算 send-dsl の定義：遷移演算 send-dsl を定義する等式は以下のとおりである．

```
op c-send-dsl : Sys Id -> Bool
eq c-send-dsl(S,I) = senderflag(S) and (sendergetid(S) = I)
                                   and (sendermsg(S) = sel) .
--
eq senderflag(send-dsl(S,I)) = senderflag(S) .
eq sendergetid(send-dsl(S,I)) = sendergetid(S) .
eq sendermsg(send-dsl(S,I)) = sendermsg(S) .
ceq iflag(send-dsl(S,I),J) = (if I = J then true else false fi)
                            if c-send-dsl(S,I) .
ceq imsg(send-dsl(S,I),J) = (if I = J then dsl else sen fi)
                           if c-send-dsl(S,I) .
ceq send-dsl(S,I) = S
                  if not c-send-dsl(S,I) .
```

send-dsl は選択した Target を解放させる時に用いる．効力条件は sender が Target を選択している時である．

遷移演算 rece-dsl の定義：遷移演算 rece-dsl を定義する等式は以下のとおりである．

```
op c-rece-dsl : Sys Id -> Bool
eq c-rece-dsl(S,I) = iflag(S,I) and (sendergetid(S) = I)
                                 and (imsg(S,I) = dsl) .
--
ceq senderflag(rece-dsl(S,I)) = false
                               if c-rece-dsl(S,I) .
ceq sendergetid(rece-dsl(S,I)) = sender
```

```
                                if c-rece-dsl(S,I) .
  ceq sendermsg(rece-dsl(S,I)) = dsl
                                if c-rece-dsl(S,I) .
  ceq iflag(rece-dsl(S,I),J) = (if I = J then false else iflag(S,J) fi)
                               if c-rece-dsl(S,I) .
  eq imsg(rece-dsl(S,I),J) = imsg(S,J) .
  ceq rece-dsl(S,I) = S
                    if not c-rece-dsl(S,I) .
```

rece-dsl により Target の解放が成功する．sender，Target 共に初期状態へ戻る．

作成した NFC の CafeOBJ 仕様は，後述する付録 A に示す．

第5章　形式化したNFC仕様の性質検証

NFC プロトコルはフィールドに複数のデバイスが存在する場合，SDD アルゴリズムによって，複数の Target から 1 つの Target を選んで通信を行う．Initiator が 1 つの Target を選択し，その Target と通信を行う際にはその Target に sel_req を送信する．Initiator が Target を選択し，解放するまで 1 つの Target と通信を行っていれば，NFC プロトコルの仕様を満たしているといえる．よって以下のようにモジュールを宣言する．

```
mod INV1{
  pr(PROTO)
  op inv1 : Sys Id Id -> Bool
  var S : Sys
  vars I J : Id
  eq inv1(S,I,J) = ((imsg(S,I) = sel) and (imsg(S,J) = sel))
                       implies (I = J) .
}
```

inv1 は証明すべき論理式であり，sel_req を受け取る Target は 1 つであるということを示している．

各帰納段階で証明すべき論理式を記述したモジュールを以下のとおりに宣言する．

```
mod ISTEP{
  pr(INV1)
  ops s s' : -> Sys
  op istep : Id Id -> Bool
  vars I J K : Id
  eq istep(I,J) = inv1(s,I,J) implies inv1(s',I,J) .
}
```

定数 s は任意の状態を示し，定数 s' は状態 s の事後状態を示す．項 inv1(s,I,J) は各帰納段階で証明すべき論理式である．

inv1 の証明譜

基底段階の証明節

基底段階 ([init]) の証明節は以下のとおりである.

```
-- [init]
open INV1
  ops i j : -> Id .
-- |=
  red inv1(init,i,j) .
close
```

この証明節に対し，CafeOBJ は true を返す.

遷移演算 send-sen に関する帰納段階

遷移演算 send-sen に関する帰納段階 ([send-sen]) について考える．効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する．CafeOBJ は前者，後者に対し，true を返す.

遷移演算 rece-sen に関する帰納段階

遷移演算 rece-sen に関する帰納段階 ([rece-sen]) について考える．効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する．CafeOBJ は前者，後者に対し，true を返す.

遷移演算 send-sdd に関する帰納段階

遷移演算 send-sdd に関する帰納段階 ([send-sdd]) について考える．効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する．CafeOBJ は前者，後者に対し，true を返す.

遷移演算 rece-sdd に関する帰納段階

遷移演算 rece-sdd に関する帰納段階 ([rece-sdd]) について考える．効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する．CafeOBJ は前者，後者に対し，true を返す.

遷移演算 send-sel に関する帰納段階

遷移演算 send-sel に関する帰納段階 ([send-sel]) について考える．効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する．CafeOBJ は前者に対し，true でも false でもない Bool の項を返し，後者に対し，true を返す.

前者の証明節を，場合分けにより分割する．以下の Bool の 2 つの項に基づいて，前者の証明節を以下のとおり 3 つに分割する.

- i = k
- j = k

分割された 3 つの場合は以下のとおりである.

1. not(i = k)
2. i = k , j = k
3. i = k , not(j = k)

1 番目から 3 番目までのいずれの証明節に対しても，CafeOBJ は true を返す.

遷移演算 rece-sel に関する帰納段階

遷移演算 rece-sel に関する帰納段階 ([rece-sel]) について考える．効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する．CafeOBJ は前者，後者に対し，true を返す.

遷移演算 send-dsl に関する帰納段階

遷移演算 send-dsl に関する帰納段階 ([send-dsl]) について考える．効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する．CafeOBJ は前者に対し，true でも false でもない Bool の項を返し，後者に対し，true を返す.

前者の証明節を，場合分けにより分割する．以下の Bool の項に基づいて，前者の証明節を以下のとおり 2 つに分割する.

- i = k

分割された 2 つの場合は以下のとおりである.

1. i = k
2. not(i = k)

いずれの証明節に対しても，CafeOBJ は true を返す.

遷移演算 rece-dsl に関する帰納段階

遷移演算 rece-dsl に関する帰納段階 ([rece-dsl]) について考える．効力条件が成り立つ場合と成り立たない場合のそれぞれについて証明節を作成する．CafeOBJ は前者，後者に対し，true を返す.

これにより，Initiator が Target を選択し，解放するまでの間，1 つの Target とのみ通信を行っていることの証明ができた.

最終的に得られた inv1 の証明譜は後述する付録 B に示す.

第6章　結論

本稿では，国際標準規格となったNFCの仕様を代数仕様記述言語CafeOBJを用いて，OTS/CafeOBJ法により，形式化し，検証を行った．NFCプロトコルでは複数のTargetからの反応を想定しており，いくつかのTargetからSDDアルゴリズムにより1つのTargetを選択する．そこで，複数のTargetから1つのTargetを選択し，そのTargetを解放するまで，そのTargetとのみ通信を行うという性質の検証を行った．NFCの仕様書の記述通りでは，1つのTargetとのみ通信を行うという性質の検証には不十分ではないかと考え，Targetを一度選択したら，解放するまでselコマンドを送らないことや，選択したTargetを解放するまでSDDアルゴリズムを始めないといった条件を加えた結果，検証をスムーズに行うことができた．それらを条件に追加することは，NFCプロトコルに基づいたシステムを作成，理解する際の支援となりえる．また，NFCの仕様にはTargetが通信の途中で切断された場合について，詳細に記述されていないので，記述の追加が必要であると考えられる．

今後の課題としては，1つのTargetとのみ通信するという性質を満たすために，NFCの仕様に最低限どのような条件を加えれば，性質を満たすのかといった限界について解析を行っていく．また，今回のCafeOBJの仕様は，NFCプロトコルをある程度抽象化しているので，その抽象度をさげ，より具体的なNFCのCafeOBJ仕様を記述し，そのCafeOBJ仕様についても検証を行っていく．

謝辞

本研究を進めるにあたり，御指導頂いた二木厚吉 教授に深く感謝致します.
また，有益な御助言をして頂いた緒方和博 特任准教授，Rene VESTERGAARD 准教授，中村正樹 助教授に御礼を申し上げます.
最後に，公私に渡り付き合って頂いた言語設計学講座の皆様に感謝致します.

参考文献

[1] 郵政省 IC カード研究会, 情報通信ネットワークで花ひらく　IC カード時代, 株式会社オーム社, 1986.

[2] 浅野正一郎, 非接触 IC カード・RFID 普及委員会 編者, 非接触 IC カード・RFID ガイドブック 2003, (株) シーメディア, 2002.

[3] 根日屋英之, 植竹古都美, ユビキタス無線工学と微細 RFID　第 2 版　－無線 IC タグの技術－, 東京電機大学出版局, 2004.

[4] klaus Finkenzeller, ソフト工学研究所 訳, RFID ハンドブック－第 2 版－　－非接触 IC カードの原理と応用－, 日刊工業新聞社, 2004.

[5] FeliCa, http://www.sony.co.jp/Products/felica/ .

[6] 社会法人日本自動認識システム協会, これでわかった RFID, 株式会社オーム社, 2003.

[7] NFC, http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-340.pdf .

[8] CafeOBJ, http://www.ldl.jaist.ac.jp/cafeobj/ , http://www.jaist.ac.jp/ kokichi/class/i636-0612/ .

[9] 二木厚吉, 緒方和博, 中村正樹, CafeOBJ 入門, コンピュータソフトウェア, 掲載予定.

[10] 緒方和博, 二木厚吉, 書き換えによるセキュリティプロトコルの帰納的検証, コンピュータソフトウェア, Vol.20, No.3(2003).

[11] Kazuhiro Ogata, Kokichi Futatsugi, Modeling and Verification of System Behavior Based on Observations., コンピュータソフトウェア, Vol.8, No.6(1991).

付録 A

```
mod* PROTO{
  pr(ID)
  pr(MSG)
  *[Sys]*

-- any initial state

  op init : -> Sys

-- observation

  bop senderflag : Sys -> Bool
  bop sendergetid : Sys -> Id
  bop sendermsg : Sys -> Msg
  bop iflag : Sys Id -> Bool
  bop imsg : Sys Id -> Msg

-- actions

  bop send-sen : Sys -> Sys
  bop rece-sen : Sys Id -> Sys
  bop send-sdd : Sys -> Sys
  bop rece-sdd : Sys Id -> Sys
  bop send-sel : Sys Id -> Sys
  bop rece-sel : Sys Id -> Sys
  bop send-dsl : Sys Id -> Sys
  bop rece-dsl : Sys Id -> Sys

-- CafeOBJ variables

  var S : Sys
```

```
  vars I J : Id
  var M : Msg

-- for any initial state

  eq senderflag(init) = false .
  eq sendergetid(init) = sender .
  eq sendermsg(init) = none .
  eq iflag(init,I) = false .
  eq imsg(init,I) = none .


-- for send-sen(Sys)

  op c-send-sen : Sys -> Bool
  eq c-send-sen(S) = not senderflag(S) .
  --
  eq senderflag(send-sen(S)) = senderflag(S) .
  eq sendergetid(send-sen(S)) = sendergetid(S) .
  eq sendermsg(send-sen(S)) = sendermsg(S) .
  ceq iflag(send-sen(S),I) = true                 if c-send-sen(S) .
  ceq imsg(send-sen(S),I) = sen                   if c-send-sen(S) .
  ceq send-sen(S) = S                             if not c-send-sen(S) .


-- for rece-sen(Sys,Id)

  op c-rece-sen : Sys Id -> Bool
  eq c-rece-sen(S,I) = iflag(S,I) and imsg(S,I) = sen and not senderflag(S) .
  --
  eq senderflag(rece-sen(S,I)) = senderflag(S) .
  eq sendergetid(rece-sen(S,I)) = sendergetid(S) .
  ceq sendermsg(rece-sen(S,I)) = sen
                               if c-rece-sen(S,I) .
  ceq iflag(rece-sen(S,I),J) = (if I = J then false else iflag(S,J) fi)
                             if c-rece-sen(S,I) .
  eq imsg(rece-sen(S,I),J) = imsg(S,J) .
  ceq rece-sen(S,I) = S
```

```
                    if not c-rece-sen(S,I) .


-- for send-sdd(Sys)

  op c-send-sdd : Sys -> Bool
  eq c-send-sdd(S) = (sendermsg(S) = sen) and not senderflag(S) .
  --
  eq senderflag(send-sdd(S)) = senderflag(S) .
  eq sendergetid(send-sdd(S)) = sendergetid(S) .
  eq sendermsg(send-sdd(S)) = sendermsg(S) .
  ceq iflag(send-sdd(S),I) = true              if c-send-sdd(S) .
  ceq imsg(send-sdd(S),I) = sdd                if c-send-sdd(S) .
  ceq send-sdd(S) = S                          if not c-send-sdd(S) .


-- for rece-sdd(Sys,Id)

  op c-rece-sdd : Sys Id -> Bool
  eq c-rece-sdd(S,I) = iflag(S,I) and (imsg(S,I) = sdd) and not senderflag(S) .
  --
  eq senderflag(rece-sdd(S,I)) = senderflag(S) .
  ceq sendergetid(rece-sdd(S,I)) = I
                                 if c-rece-sdd(S,I) .
  ceq sendermsg(rece-sdd(S,I)) = sdd
                               if c-rece-sdd(S,I) .
  ceq iflag(rece-sdd(S,I),J) = (if I = J then false else iflag(S,J) fi)
                             if c-rece-sdd(S,J) .
  eq imsg(rece-sdd(S,I),J) = imsg(S,J) .
  ceq rece-sdd(S,I) = S
                    if not c-rece-sdd(S,I) .


-- for send-sel(Sys,Id)

  op c-send-sel : Sys Id -> Bool
  eq c-send-sel(S,I) = (not senderflag(S)) and (sendergetid(S) = I)
                                           and (sendermsg(S) = sdd) .
```

```
  --
  ceq senderflag(send-sel(S,I)) = true
                               if c-send-sel(S,I) .
  eq sendergetid(send-sel(S,I)) = sendergetid(S) .
  eq sendermsg(send-sel(S,I)) = sendermsg(S) .
  ceq iflag(send-sel(S,I),J) = (if I = J then true else false fi)
                            if c-send-sel(S,I) .
  ceq imsg(send-sel(S,I),J) = (if I = J then sel else sen fi)
                           if c-send-sel(S,I) .
  ceq send-sel(S,I) = S
                   if not c-send-sel(S,I) .


-- for rece-sel(Sys,Id)

  op c-rece-sel : Sys Id -> Bool
  eq c-rece-sel(S,I) = iflag(S,I) and (imsg(S,I) = sel) and sendergetid(S) = I .
  --
  eq senderflag(rece-sel(S,I)) = senderflag(S) .
  eq sendergetid(rece-sel(S,I)) = sendergetid(S) .
  ceq sendermsg(rece-sel(S,I)) = sel
                              if c-rece-sel(S,I) .
  ceq iflag(rece-sel(S,I),J) = (if I = J then false else iflag(S,J) fi)
                            if c-rece-sel(S,I) .
  eq imsg(rece-sel(S,I),J) = imsg(S,J) .
  ceq rece-sel(S,I) = S
                   if not c-rece-sel(S,I) .


-- for send-dsl(Sys,Id)

  op c-send-dsl : Sys Id -> Bool
  eq c-send-dsl(S,I) = senderflag(S) and (sendergetid(S) = I)
                                     and (sendermsg(S) = sel) .
  --
  eq senderflag(send-dsl(S,I)) = senderflag(S) .
  eq sendergetid(send-dsl(S,I)) = sendergetid(S) .
  eq sendermsg(send-dsl(S,I)) = sendermsg(S) .
```

```
  ceq iflag(send-dsl(S,I),J) = (if I = J then true else false fi)
                              if c-send-dsl(S,I) .
  ceq imsg(send-dsl(S,I),J) = (if I = J then dsl else sen fi)
                             if c-send-dsl(S,I) .
  ceq send-dsl(S,I) = S
                   if not c-send-dsl(S,I) .


-- for rece-dsl(Sys,Id)

  op c-rece-dsl : Sys Id -> Bool
  eq c-rece-dsl(S,I) = iflag(S,I) and (sendergetid(S) = I)
                                   and (imsg(S,I) = dsl) .
  --
  ceq senderflag(rece-dsl(S,I)) = false
                                if c-rece-dsl(S,I) .
  ceq sendergetid(rece-dsl(S,I)) = sender
                                 if c-rece-dsl(S,I) .
  ceq sendermsg(rece-dsl(S,I)) = dsl
                               if c-rece-dsl(S,I) .
  ceq iflag(rece-dsl(S,I),J) = (if I = J then false else iflag(S,J) fi)
                              if c-rece-dsl(S,I) .
  eq imsg(rece-dsl(S,I),J) = imsg(S,J) .
  ceq rece-dsl(S,I) = S
                   if not c-rece-dsl(S,I) .

}
```

付録B

```
-- [init]
open INV1
  ops i j : -> Id .
-- |=
  red inv1(init,i,j) .
close


-- [send-sen(s)]
-- [c-send-sen]
open ISTEP
  ops i j : -> Id .
--  eq c-send-sen(s) = true .
  eq senderflag(s) = false .
  eq s' = send-sen(s) .
-- |=
  red istep(i,j) .
close

-- [send-sen(s)]
-- [^c-send-sen]
open ISTEP
  ops i j : -> Id .
--  eq c-send-sen(s) = false .
  eq senderflag(s) = true .
  eq s' = send-sen(s) .
-- |=
  red istep(i,j) .
close
```

```
-- [rece-sen(s,k)]
-- [c-rece-sen]
open ISTEP
  ops i j k : -> Id .
--  eq c-rece-sen(s,k) = true .
  eq iflag(s,k) = true .
  eq imsg(s,k) = sen .
  eq senderflag(s) = false .
  eq s' = rece-sen(s,k) .
-- |=
  red istep(i,j) .
close

-- [rece-sen(s,k)]
-- [^c-rece-sen]
open ISTEP
  ops i j k : -> Id .
  eq c-rece-sen(s,k) = false .
  eq s' = rece-sen(s,k) .
-- |=
  red istep(i,j) .
close


-- [send-sdd(s)]
-- [c-send-sdd]
open ISTEP
  ops i j : -> Id .
--  eq c-send-sdd(s) = true .
  eq sendermsg(s) = sen .
  eq senderflag(s) = false .
  eq s' = send-sdd(s) .
-- |=
  red istep(i,j) .
close

-- [send-sdd(s)]
-- [^c-send-sdd]
```

```
open ISTEP
  ops i j : -> Id .
  eq c-send-sdd(s) = false .
  eq s' = send-sdd(s) .
-- |=
  red istep(i,j) .
close


-- [rece-sdd(s,k)]
-- [c-rece-sdd]
open ISTEP
  ops i j k : -> Id .
--  eq c-rece-sdd(s,k) = true .
  eq iflag(s,k) = true .
  eq imsg(s,k) = sdd .
  eq senderflag(s) = false .
  eq s' = rece-sdd(s,k) .
-- |=
  red istep(i,j) .
close

-- [rece-sdd(s,k)]
-- [^c-rece-sdd]
open ISTEP
  ops i j k : -> Id .
  eq c-rece-sdd(s,k) = false .
  eq s' = rece-sdd(s,k) .
-- |=
  red istep(i,j) .
close


-- [send-sel(s,k)]
-- [c-send-sel,i=k,j=k]
open ISTEP
  ops i j k : -> Id .
--  eq c-send-sel(s,k) = true .
```

```
  eq senderflag(s) = false .
  eq sendergetid(s) = k .
  eq sendermsg(s) = sdd .
  eq i = k .
  eq j = k .
  eq s' = send-sel(s,k) .
-- |=
  red istep(i,j) .
close

-- [send-sel(s,k)]
-- [c-send-sel,i=k,^j=k]
open ISTEP
  ops i j k : -> Id .
--  eq c-send-sel(s,k) = true .
  eq senderflag(s) = false .
  eq sendergetid(s) = k .
  eq sendermsg(s) = sdd .
  eq i = k .
  eq (j = k) = false .
  eq s' = send-sel(s,k) .
-- |=
  red istep(i,j) .
close

-- [send-sel(s,k)]
-- [c-send-sel,^i=k]
open ISTEP
  ops i j k : -> Id .
--  eq c-send-sel(s,k) = true .
  eq senderflag(s) = false .
  eq sendergetid(s) = k .
  eq sendermsg(s) = sdd .
  eq (i = k) = false .
  eq s' = send-sel(s,k) .
-- |=
  red istep(i,j) .
close
```

```
-- [send-sel(s,k)]
-- [^c-send-sel]
open ISTEP
  ops i j k : -> Id .
  eq c-send-sel(s,k) = false .
  eq s' = send-sel(s,k) .
-- |=
  red istep(i,j) .
close


-- [rece-sel(s,k)]
-- [c-rece-sel]
open ISTEP
  ops i j k : -> Id .
--  eq c-rece-sel(s,k) = true .
  eq iflag(s,k) = true .
  eq imsg(s,k) = sel .
  eq sendergetid(s) = k .
  eq s' = rece-sel(s,k) .
-- |=
  red istep(i,j) .
close

-- [rece-sel(s,k)]
-- [^c-rece-sel]
open ISTEP
  ops i j k : -> Id .
  eq c-rece-sel(s,k) = false .
  eq s' = rece-sel(s,k) .
-- |=
  red istep(i,j) .
close


-- [send-dsl(s,k)]
-- [c-send-dsl,i=k]
```

```
open ISTEP
  ops i j k : -> Id .
--  eq c-send-dsl(s,k) = true .
  eq senderflag(s) = true .
  eq sendergetid(s) = k .
  eq sendermsg(s) = sel .
  eq i = k .
  eq s' = send-dsl(s,k) .
-- |=
  red istep(i,j) .
close

-- [send-dsl(s,k)]
-- [c-send-dsl,^i=k]*
open ISTEP
  ops i j k : -> Id .
--  eq c-send-dsl(s,k) = true .
  eq senderflag(s) = true .
  eq sendergetid(s) = k .
  eq sendermsg(s) = sel .
  eq (i = k) = false .
  eq s' = send-dsl(s,k) .
-- |=
  red istep(i,j) .
close

-- [send-dsl(s,k)]
-- [^c-send-dsl]
open ISTEP
  ops i j k : -> Id .
  eq c-send-dsl(s,k) = false .
  eq s' = send-dsl(s,k) .
-- |=
  red istep(i,j) .
close


-- [rece-dsl(s,k)]
```

```
-- [c-rece-dsl]
open ISTEP
  ops i j k : -> Id .
--  eq c-rece-dsl(s,k) = true .
  eq iflag(s,k) = true .
  eq sendergetid(s) = k .
  eq imsg(s,k) = sel .
  eq s' = rece-dsl(s,k) .
-- |=
  red istep(i,j) .
close

-- [rece-dsl(s,k)]
-- [^c-rece-dsl]
open ISTEP
  ops i j k : -> Id .
  eq c-rece-dsl(s,k) = false .
  eq s' = rece-dsl(s,k) .
-- |=
  red istep(i,j) .
close
```