ソフトウェア工学 89−18
（１９９２．１２．１５）

SDLとLOTOSの比較

安藤津芳† 太田正孝† 高橋薫‡

†高度通信システム研究所 ‡東北大学

本報告では，通信システム用の仕様記述言語であるCCITT勧告のSDLとISO規格のLOTOSを多面的に比較する．具体的には，まずSDLとLOTOSについて，一般的な比較をそのモデルと記述について行う．次に，仕様の具体化と検証を目的に，両仕様記述言語の相互解釈の可能性について検討する．最後に，設計における仕様化と仕様の具体化について両仕様記述言語を用いた場合での違いをまとめる．

SDL Vs. LOTOS

Tsuyoshi Ando† Masataka Ohta† Kaoru Takahashi‡

†AIC System Laboratories

6-6-3,MinamiYoshinari,Aoba-ku,Sendai-shi, Miyagi,989-32 JAPAN

‡Tohoku University

2-1-1,Katahira,Aoba-ku,Sendai-shi,Miyagi,980 JAPAN

ABSTRACT In this paper, we compare SDL and LOTOS from several points of view. They are formal description languages for communication systems specifications. SDL is recommended by CCITT, and LOTOS is standardized by ISO. First, we compare their model and description. Next, we discuss two-way interpretation of their languages. Finally, in design step and implementation step, we discuss methodology of each language.

1. はじめに

近年，ソフトウェアの仕様を形式的仕様記述言語を用いて定義することで自動プログラミングや仕様検証と言った機械支援を期待すると共に，仕様の曖昧性の除去･欠落の防止を望む傾向が強くなっている．これに伴い通信ソフトウェアにおいてもCCITT, ISOで形式的仕様記述言語が勧告されている[(1),(2),(3),(4),(5)]．しかし，その発生起源の違いからCCITT勧告のSDLは主に交換ソフトウェアの設計で使用され，ISO勧告のEstelle, LOTOSは主にコンピュータソフトウェア，なかでもプロトコルの設計で使用されている．加えて，これらの仕様記述言語によって両機構から出されている勧告のいくつかが形式的記述により定義されているか，されようとしている．具体的にはCCITTでは交換機のサービス定義にSDLが用いられているし，ISOではOSIの各レイヤのプロトコルやサービスの定義，更にはODP(開放型分散処理)での参照モデルの定義に使用されようとしている．このように，通信ソフトウェアの設計者にとって直接的･間接的に仕様記述言語に触れる機会が増えてきており，これらの仕様記述言語の特徴を知ることは，設計作業を進める上で有益になる．

そこで，本報告では，これらの仕様記述言語のうちCCITTのSDLとISOのLOTOSを多面的に比較･評価する．ここで，ISOの規格からLOTOSを選択した理由は，EstelleはSDLとその基になっているモデルが類似していることと，CCITTで以前，SDLの次の仕様記述言語としてLOTOSを注目していたことによる．本報告では，まずSDLとLOTOSについて，一般的な比較をそのモデルと記述について行う．次に，両仕様記述言語の相互解釈の可能性について整理する．そして，設計における仕様化と仕様の具体化について両仕様記述言語での違いをまとめる．

2. SDLとLOTOSの一般的比較

本章では，SDLとLOTOSの両仕様記述言語が基盤としているモデルと記述について一般的な比較を行う．まず，両仕様記述言語の簡単な説明をする．

2.1 概論

SDLは，非同期通信プロセスによる状態遷移モデルを基盤として，設計･製造工程における仕様化を対象に勧告化されている．SDLの長所はその理解性や記述性にあり，設計･製造工程における仕様定義･検討･整理資料としてだけでなく保守ドキュメントとしても有効に使用されている．しかし，一方ではSDL自体の持つ仕様検証能力が乏しいために，仕様検証の立場からは，現状は自動プログラミングの結果を用いたシミュレーションによる動作確認が中心になっており，仕様確認の作業効率の向上が課題である[(6)]．これに対しての従来の研究では，個々の仕様のデッドロックやライブロック･到達可能性を調べるといったValidation指向の検証方法の提案[(6),(8),(9),(10)]が中心である．

一方，LOTOSは，同期通信プロセスによる時系列表記を基盤として，要求の仕様化を対象に標準化されている．その特徴としては，理解性や記述性ではSDLに劣るものの，仕様検証の立場からは，デッドロックやライブロック･到達可能性を調べるといったValidation機能はもちろんのこと，複数の仕様間の関係(同値関係，包含関係等)を調べるVerification機能をも持っている[(2),(11),(12),(15)]．例えば，CCITTの勧告している交換サービスがLOTOSで仕様記述されていたとした場合に，設計者がLOTOSを用いて対象交換システムのサービスまたはソフトウェアの仕様を記述すれば，その二つの仕様の間の関係を調べることができる．また，機能追加時に新しい仕様が元になっている仕様を満足しているかどうかを調べるといったVerificationが可能となる．

2.2 モデル

ここでは，SDLとLOTOSそれぞれが基本としているモデルについて比較する．

SDLでは，通信処理を実行する単位をProcessと呼び，一般に記述対象のシステムを複数のProcessから構成する．各Processは，拡張有限状態機械としてモデル化され，各Processとの間で信号を送受することにより，その入力情報と元の状態に応じて必要な処理を実行し次の状態に遷移する．この信号送受において，各Processは任意の時点で他Processに信号を送出することが可能でその後必要な処理を継続して実行することができる．このため，各Processは容量が十分大きなFIFO(First In First Out)型の信号受信バッファを1個持っており，他Processから受け取る信号はすべてこのバッファを経由して入力する．[(6)]

LOTOSではシステムの記述を「Process」の観点から行い，外部から観測される振る舞いとしての通信能力を記述することでシステムの仕様記述を行う．「Process」とは，その環境(対象としているProcessの外側全体)における他のProcessと通信を行う抽象的な実体を示す．Process間の通信は「Gate」と呼ばれる作用点で起こる．インタラクション(通信による相互作用)の基本単位は「アクション」と呼ばれ，Gateとデータの値の組からなる．たとえば，あるGateでのあるデータ値の交換(送信や受信)が一つのアクションに対応する．アクションは「アトミック」で「同期的」なインタラクションである．アクションがアトミックであるというのは，(1)瞬間的であり，(2)細分化できず，(3)他のアクションと時間的な重なりがない，ということを意味する．アクションが同期的であるというのは環境も含めて，通信する二つ以上のProcessが，同時にその発生に関与するということを表す．[(7)]

以上のように，SDLとLOTOSをモデルレベルで比較すると，SDLは複数の拡張有限状態機械(Process)が非同期通信を用いて情報のやり取りをしながら処理を実現するモデルであるのに対して，LOTOSはProcessの外部から観測可能なイベント間の時間順序を規定することを目的に仕様化され，イベントは同期通信により実現されるモデルである．

2.3 記述

それぞれの仕様記述言語を用いた場合のユーザによる仕様の書き方･スタイルについて，仕様構造と動作記述，仕様化対象レベルの観点より比較する．

2.3.1 仕様構造

SDLで記述する仕様の構造はSystem, Block, Processを基本とする．ここで，Systemは仕様化対象

の全体を表し，BlockはSystemを機能的に分割した単位であり，Block内に定義されるProcessが実際の動作を表す．従って，SDLにおけるBlockは仕様の静的な分割を表すものである．また，高階層な仕様記述を行う場合にはSubstructureを導入することで，Blockを何階層にも分割できる．そして，この階層のレベルに対応して信号送受のための路が，Block間ではChannel，Process間ではSignalrouteとして定義される．さらに，Processの特性として，Systemの初期生成時より動作可能なものと，他Processインスタンスから生成されて動作可能になるものを，分けて定義することができる．

一方，LOTOSは，通信動作を定義するProcessを基本とし，仕様化対象の全体を表すProcessを特別にSpecificationと呼ぶ．Processは，機能的にも処理的にも何階層にも分割が可能であるが，各階位のそれぞれのProcessに動作が記述できるため，仕様化対象システムの静的な分割を表すものではない．また，Specificationを除くすべてのProcessは，Specificationまたは他Processインスタンスから生成されて初めて動作可能になる．

以上より，SDLとLOTOSの仕様構造は，SDLが静的な仕様化対象システムの構造の定義と動的な動作仕様の定義機能を併せて持っているのに対して，LOTOSは動的な動作仕様の定義機能が主であるといえる．

2.3.2 動作の記述

動作の記述を比較する上で，各仕様記述言語の持つ記述スタイルと動作表現に分けて整理する．

(1) 記述スタイル

SDLの記述スタイルは，SDLの動作定義の基本となるProcessが，拡張有限状態機械に基づいているため，状態遷移(指向)スタイルに統一されている．

一方，LOTOSの記述スタイルは，文献(13)にあるように，外部から観測可能なイベントを中心としたもの(Extensionalスタイル)と，内部処理を中心としたもの(Intentionalスタイル)の，二種類に大きくわかれる．そしてそれぞれが表1のような記述スタイルを持つ．

表1. LOTOSの記述スタイル一覧

スタイル名		記述の特徴・方法
Extensional スタイル	モノリシックスタイル	システムの外部的な挙動を，構造を持たせずに記述
	制約指向スタイル	システムの外部的な挙動を，種々の個別的な制約の集まりとして記述
Intentional スタイル	状態指向スタイル	システムの状態を記述上に陽に反映させて記述
	リソース指向スタイル	システムの構成要素を記述上に反映させて記述

このように，記述スタイルという点に着目してみるとLOTOSがブラックボックス型仕様記述とホワイトボックス型仕様記述の両方に対応しているのに対して，SDLはホワイトボックス型仕様記述に対応している．このことから，LOTOSはSDLの記述スタイルを包含しているといえる．

(2) 動作表現

ここでは，SDLとLOTOSの持つ動作表現について比較する．

SDLの持つ動作表現は，Process間の信号送受に関するもの，時間制御(Timer)に関するもの，内部処理に関するもの，状態遷移に関するもの，Process生成に関するものに分類できる．それぞれについて表2に示す．

表2. SDLの動作表現の分類

項目名	機能概要	SDLの処理
Process間信号送受関連	Process間の通信やデータのやり取りを制御	Input, Output, Save等
時間制御(Timer)関連	各Processにおける時間的制約を明示的に制御	Set, Reset等
内部処理関連	各Processにおける単純な処理の記述と処理の制御	Task, Decision, Join, Label, Call, Return等
状態遷移関連	各Processにおける状態を制御	Start, State, Nextstate
Process生成関連	Processの生成と消滅を制御	Create, Stop等

一方，LOTOSの持つ動作表現は，Process間の信号送受に関するもの，内部処理に関するもの，Process生成に関するものに分類できる．それぞれについて表3に示す．

表3. LOTOSの動作表現の分類

項目名	機能概要	LOTOSの処理
Process間信号送受関連	Process間の通信やデータのやり取りを制御	Gateと同期条件
内部処理関連	内部処理そのもの(i)と内部処理の制御	choice, i, enable, disable
Process生成関連	Processの生成と消滅を制御	Processのインスタンシエーション，exit, stop

以上より，SDLの方がLOTOSより内部処理の制御記述を重視していることが，その動作表現の多さから分かる．

2.3.3 仕様化対象レベル

SDLの仕様化対象レベルは，その名前の示すようにSpecificationとDescriptionであるが，ここでのSpecificationは，概要設計仕様(交換ソフトウェアにおけるサービス仕様)に対応しDescriptionが詳細設計仕様に対応する．これに対してLOTOSはSpecificationを対象としているが，このSpecificationとは，対象システムの要求仕様と設計仕様に対応する．このことはLOTOSの持つ記述スタイルや動作表現からの帰結である．

基本的にこれらの違いが有るが，これはLOTOSが要求仕様記述指向で設計されているのに対して，

SDLが設計仕様記述指向で設計されていることに基づくものである.

3. 相互解釈

前章の帰結から, 一般には, 要求仕様をLOTOSで記述し, その詳細化の結果である設計仕様またはプログラム仕様をSDLで記述することが考えられる. また, これとは逆にSDLで記述された仕様をLOTOSに解釈して検証に用いることも考えられる. そこで, 本章では, この両仕様記述言語で記述された仕様の相互解釈の方法とその可能性を述べる.

3.1 LOTOS機能のSDL解釈

まず, LOTOSの機能を, SDLでどのように解釈できるか考える. LOTOSの表現の中でSDLに解釈する場合の主な項目としては次のものが挙げられる.

- 多重同期通信機構の解釈
- 階層構造の解釈
- データ定義の解釈
- gate定義の解釈
- キーワード対応の解釈

これらに対する基本的な考え方を次に示す. 多重同期通信機構は, SDLの共有変数定義であるView/Reveal及びExport/Import定義を用いて, 同期の基本処理を実現する. もちろんこれだけですべての多重同期通信が解釈できるわけではないので, 解釈の対象となるLOTOS記述には制約が必要である. 次に階層定義の解釈は, SDLの上位階層(System,Block)では, 処理が記述できないので, このようなLOTOS仕様は対象外とすれば比較的簡単に対応は付けられる. データの定義は, LOTOS, SDL共に同じ抽象型データ定義言語ACTONEに基づいている. しかし, LOTOSのEquationsを理解してSDLのAxiomsに解釈することは一般的には困難である. そこで, 事前に用意したSDLのデータ定義に対応するLOTOS定義と, 双方で同様に使用できるLOTOS定義に対象を制限する必要が有る. gate定義の解釈は, 各gateをSDLの信号路定義(Channel,Signalroute)とその接続定義(Connect)に変換すると共にその同期通信を実現するための処理機構へ連結する処理へ解釈する必要が有る. この解釈も一般的対応は困難であるので, なんらかの制約が必要であろう. キーワード対応の解釈は, 簡単に対応づけできるものとそうでないものに別れる. 即ち, disableやstopの解釈は簡単にできるがそれ以外のものに関しては. かなり高度な解釈が必要になると考えられる. 以上をまとめて表4に示す.

この表からも判るように, LOTOS仕様のSDL仕様への機械解釈は, かなり厳しいものが有る. しかし, これまでにLOTOSのC言語へのコンパイラが存在する[14]ことを考えれば, 種々の制約は付くかもしれないが, 実現不可能ではない.

前章で述べているように, LOTOSとSDLでは, LOTOS仕様のほうが前工程に現れるものであるので, この方向の解釈は重要となる.

3.2 SDL機能のLOTOS解釈

次に, 前節とは逆にSDLの機能をどのようにLOTOS解釈する[16]かを考える. SDLの表現の中で

表4. LOTOS機能のSDL解釈案

No.	LOTOSの表現/機能	SDL解釈案	機械解釈性
1	多重同期通信機構	データ参照による同期制御.	制約要
2	階層化記述	SDLの階層化に対応	制約要
3	**specification**定義	System定義	容易
4	gate定義	信号路とその接続関係及び同期データの定義	制約要
5	specification parameter定義	External Syntype定義	容易
6	specification functionality	解釈せず	容易
7	**library**定義	Predefined data+ α	容易
8	**type**定義	Newtype, Syntype, Generator	困難
9	**process**	Block, Substructure, Process, Procedure	制約要
10	**let**文	データ処理	容易
11	**choice**文	Process処理	制約要
12	**par**文	並列Process処理	容易
13	**hide**文	内部Channel/Signalroute	制約要
14	enabling(>>)	解釈せず, Nextstate	制約要
15	**accept in**	Procedure return, Input	制約要
16	disabling([>)	State *;	容易
17	[]	Decision/マルチInput	制約要
18	guard expression	Decisionのアンサ部	制約要
19	**stop**	Stop	容易
20	**exit**	Return, Stop	制約要
21	!data	Output	制約要
22	?data	Input	制約要
23	guard	Decision文	制約要
24	inner action i	解釈せず/informal text	容易

LOTOSに解釈するための項目として主に次のものが挙げられる.

- 非同期通信機構の解釈
- 階層構造の解釈
- 時間制御の解釈
- 信号及び信号路の解釈
- SDLの暗黙的処理の解釈
- データ定義の解釈
- キーワード対応の解釈

これらに対する基本的な考え方を次に示す．非同期通信機構をLOTOSの同期通信で実現するには，通信をキュー経由の処理にすることで簡単に実現できる．次に，階層構造の解釈は，SDLの持つ階層構造をLOTOSのspecificationとprocessで対応させれば簡単に解釈できる．信号及び信号路の解釈は，信号はそれらを定義するためのtype定義のひな型を与えておくことで解釈は簡単にできるし，信号路の解釈に関しても各信号路及びその接続定義をLOTOSのprocessに変換することで処理を実現することができる．この時，LOTOSのgateに解釈されるのは，SDLの構造定義要素(Process等)と信号路の接点と，信号路と接続定義の接点である．SDLの暗黙的処理の解釈で対象とするのは，各Processにおける信号の扱いの部分である．即ち，「SDLの各Processは受信のためのキューを一つ持っている」とか「不要信号は破棄される」といったものである．これらの解釈はSDLの各Processを信号受信部と処理部に分割してLOTOSのprocessに解釈することと，処理部で明示的に破棄の処理を追加すればよい．データ定義の解釈は，LOTOSからSDLへの解釈の時と同様に行う．従ってこの解釈には制限が必要である．キーワード対応の解釈は主なものがLOTOSの単語または文で対応づけできる．以上をまとめて表5に示す．

この表からも判るように，先ほどのLOTOS機能のSDL解釈よりも，機械解釈の可能性はずっと高い．

この方向の解釈は，新に作成したLOTOS仕様が既にSDLで作成された仕様を満足するものかどうかの判定に用いることができる．具体的には，CCITTで勧告されている通信のサービスやプロトコルではSDLを用いているものが多い．そこで，作成するシステムのLOTOS仕様がこれらの勧告を満たすものであるかということを，この解釈を用いることで知ることができる．

4. 仕様化と具体化

本章では，SDLとLOTOSそれぞれを用いた場合の仕様化と具体化段階での違いについて考察する．

4.1 段階的詳細化への適用性

LOTOS, SDL共に，階層化記述を可能にしているが，その使用法は微妙に異なる．そこで，本節では，設計で必須となる段階的詳細化を実現する上で，それぞれの言語の階層化記述の違いから来る制約と，その有効性について整理する．

整理の方法として，段階的詳細化を3つのフェーズに分け，それぞれにおける特徴を示す．ここで，3つのフェーズとは，いわゆる機能分割に対応する処理単位の分割・詳細化のフェーズ，一旦処理単位の分割を終えて処理概要を記述したものを更に詳細化するフェーズ，処理単位の分割と処理の具体化を並行して行っていく複合フェーズである．

(1)処理単位の分割化機能

処理単位の分割・詳細化機能は，LOTOS, SDLのどちらもサポートしているが，その詳細において次のような違いが有る．

① LOTOSでの最少階層数が1に対して，SDLでは3である．

表5. SDL機能のLOTOS解釈案

No.	Basic SDLの表現/機能	LOTOS解釈案	機械解釈性
1	System定義	specification定義	容易
2	Block定義	process定義	容易
3	Channel定義	process定義	容易
4	Substructure定義	process定義	容易
5	Process定義	process定義	容易
6	Signalroute定義	process定義	容易
7	Connect定義	process定義	容易
8	Procedure定義	process定義	容易
9	上記構成要素定義の接続点	gate定義	容易
10	Signal定義	type定義	容易
11	非同期通信機能	Queue処理	容易
12	Timer関連	Predefined timer meta-process	比較的容易
13	Newtype等の形宣言	type宣言	制約要
14	Dcl(データ宣言)	processパラメータ	困難
15	Start	LOTOS上では省略	容易
16	State / Nextstate	State process / instantiation	容易
17	Input / Output	gate通信(?/!)	制約要
18	Task	type operation	困難
19	Decision	choice	容易
20	Procedure Call	process instantiation	容易
21	Return	exit(return-value)	容易
22	Create	process instantiation	容易
23	Stop	stop	容易
24	Label	process分割	制約要
26	Join	process instantiation	制約要
27	Save	Signal save processによる処理	容易
28	View / Revealed処理	Revealed data管理 processとのgate通信	容易

② LOTOSでは子孫の詳細化に制約を与えないが，SDLは与える．

③ 階層化を行った場合の，型定義やプロセス呼出のスコープが異なる．

これらを簡単に説明すると，①は，LOTOSではspecificationという最上位のレベルの記述だけでも仕様は成り立つが，SDLでは，最小でもSystem,

Block, Processといった3階層が必要であるということである．もちろんSDLにおいてこれらの三層の関係を1:1:1に対応させれば，1階層と同じことが記述できるが，厳密に言えばこれらは異なっている．このことは，最も抽象的なレベルでの仕様の記述と理解性に影響を与えるものである．

次に，②は，図1のような親子関係を持つ仕様を記述しようとした場合の違いについて述べている．

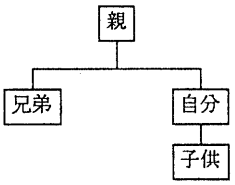


図1. 階層化における記述の異なる例

すなわち，LOTOSではこのような関係をそのまま階層化記述できるのに対して，SDLでは，Service, またはProcedureを用いれば記述可能では有るが，この場合にも制約がつく，逆にそれらを使わないのであればシンタックス上の問題が有るので，ダミーのBlock定義を"兄弟"の所に追加する必要が有る．これは，設計者の階層化手順に制約を与えるものになる．

さらに，③は，SDLが直系先祖と子供を基本としたスコープを持つのに対して，LOTOSは，これに伯父・叔母もスコープに含めるといった違いがある．このことは，LOTOSがより融通のきくスコープを持っているといえるが，しかし，一般のプログラム言語と異なるスコープであるので，設計者に誤解を招く危険を持っている．

以上をまとめると，処理単位の分割・詳細化機能としては，LOTOSのほうが設計者の考えをすなおに仕様化できる機構を備えてはいるが，自由度が高いので設計者間による記述の不統一を生む危険性を持っているといえる．

(2)処理の詳細化機能

ここでは，一旦記述した処理を更に詳細化する場合の違いについて述べる．LOTOSでは，specificationとprocessのどちらにおいても処理の記述が可能である．一方，SDLでは処理記述の可能なものはProcess, Procedure, Serviceであるが，ProcedureとServiceは，その使い方に制約を受ける．そこで，ここでは，LOTOSのprocessとSDLのProcessを比較することとする．

ここで，図2(a)に示すような処理を行うプロセスを，更に詳細化する場合(図2(b)参照)を考える．

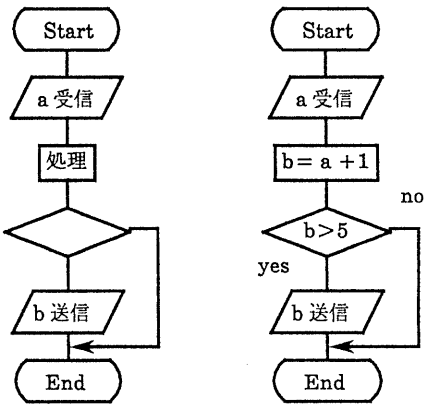
この例をLOTOSで記述した場合，次の様になる．

(a)概要処理

```
process example[gate]:noexit;=
    gate?a:nat; i; ( gate!b;stop [] i; stop )
endproc
```

(b)詳細処理

```
process example[gate]:noexit;=
    gate?a:nat;
        (  [(a+1) > 5] → gate!(a+1);stop
           []
           [(a+1) ≦ 5] → i; stop )
endproc
```



(a)概要処理　(b)詳細処理

図2. 比較のための処理プロセス詳細化例

この例では，LOTOSの内部処理表記(i)と非決定的な分岐(choice)を用いて概要処理を表しているのに対して，詳細処理では，それらをデータ型のオペレーションとガード式を用いることで具体化できる．なお，詳細処理で残っている内部処理(i)は，LOTOSの意味論上の制約から来るものである．従って，概要処理定義内では内部処理の使い方の違いを整理する必要が有る．

次に，SDLの場合を考える．同じようにこの例を記述した場合には次のようになる．

(a)概要処理

```
Process example(1,1);
    Start;
        NextState st1;
    State st1;
        Input sig(a);
            Task 'syori';
            Decision ('Bunki');
                ('case1'):   Output sig(b); Stop;
                ('case2'):   Stop;
            EndDecision;
EndProcess example;
```

(b)詳細処理

```
Process example(1,1);
    Start;
        NextState st1;
    State st1;
        Input sig(a);
            Task b := a+1;
            Decision (b>5);
                (True):    Output sig(b); Stop;
                (False):   Stop;
            EndDecision;
EndProcess example;
```

この例では，SDLのインフォーマルテキストを概要処理で用いて，詳細処理ではそれを具体化するという方法を取っている．

以上から判るように，どちらの言語でも処理の詳細化は可能であるがそれぞれ次のような特徴が有る．LOTOSは，「システムは，そのシステムとやり取りする外部から観測可能なイベント間の時間順序を規定することにより仕様化可能である」という概

念に基づいており，外部とのイベントを除いた部分の記述は内部処理表記(i)とtype定義内のオペレーションに頼っている．一方，SDLでは，インフォーマルテキストを許しているので，上位仕様でインフォーマルに定義していたものをTaskやDecisionなどを用いて具体化することができる．

(3)複合的詳細機能

ここでは，先の二種類の詳細化を，同時に行う場合の両者の違いと，設計時の考慮事項を示す．

今，図3に示したシステム(Example)の処理シーケンスを，図4に示すように詳細化することを考える．

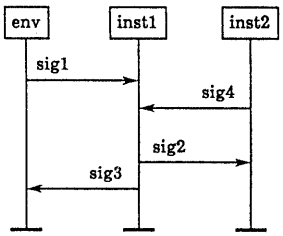
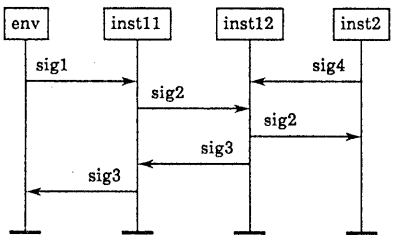


図3. 概要シーケンス



注)ここで，inst11とinst12はinst1の内部処理

図4. 詳細化シーケンス

まず，図3のシーケンスを満たす仕様をLOTOSとSDLで記述し，それらを図4の様にinst1を詳細化することを考える．すると，LOTOSでは，次のようになり，図3で規定した信号の順序を保証した形でも分割はすなおに行える．

```
specification example[env_inst1]:noexit
  type message_id is
    sorts message_id
    opns sig1,sig2,sig3,sig4 : → message_id
  endtype
  behaviour
    hide inst1_inst2 in
      inst1[env_inst1,inst1_inst2]
      |[inst1_inst2]|
      inst2[inst1_inst2]
    where
      process inst1[env_inst1,inst1_inst2]:noexit:=
        hide self_inst11,self_inst12,inst11_inst12 in
          inst11[self_inst11,inst11_inst12]
          |[self_inst11,inst11_inst12]|
          inst12[self_inst12,inst11_inst12]
          |[self_inst12]|
          ( env_inst1?x:message_id[x=sig1];
            self_inst11!x;
            inst1_inst2?y:message_id[y=sig4];
            self_inst12!y;self_inst12?z:message_id;
            inst1_inst2!z;self_inst11?z1:message_id;
            env_inst1!z1; stop )
        where
          process inst11[g1,g2]:noexit:=
            g1?x:message_id;g2!sig2;
            g2?y:message_id;g1!sig3;stop
          endproc
          process inst12[g1,g2]:noexit:=
            g1?x:message_id;g2?y:message_id;
            g1!sig2;g2!sig3;stop
          endproc
      endproc
      process inst2[inst1_inst2]:noexit:=
        inst1_inst2!sig4;inst1_inst2?x:message_id;
        stop
      endproc
endspec
```

一方，SDLでは，図3で規定した信号の順序を保証しようとすると，inst1の処理に対応するPROCESSをinst11,inst12と同じレベルに定義しないと実現できない．即ち，プロセスの構造が図5のようになる．こ

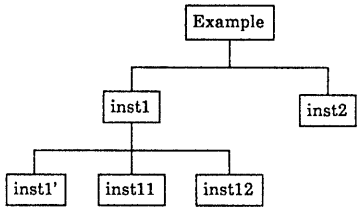


図5. SDLで詳細化した場合のProcess構造

のことは，設計者の詳細化における素直な手順を損なうだけでなく，上位で定義した仕様の流用度を下げ，誤った詳細化を行う危険を持っている．加えて，設計者が図4だけに着目して仕様記述した場合にはsig1とsig4の順番が非決定的なものになり，図3のシーケンスを満たさない仕様を作成することが有るので注意が必要である．

以上をまとめると，SDLでは，処理を含む詳細化を行う場合には，処理部分の流用性が低くなるか，すなおな階層の表現ができなくなる危険を持つのに対して，LOTOSでは，記述スタイルにも左右されるが，処理部分の高い流用性を伴って，すなおな詳細化が可能である．すなわち，SDLでは非連続的な詳細化に限定されるのに対して，LOTOSでは連続的な詳細化が可能であるといえる．

4.2 仕様からの具体化

本節では，両仕様記述言語を用いて仕様を作成した場合の，プログラムへの具体化のしやすさについて比較する．比較の立場として，人による具体化と機械による具体化の二種類を考える．また，それぞれの仕様を具体化する場合のプログラム言語についても簡単に触れる．

(1)人による具体化

人による具体化を考える上で一番重要となるのは仕様の理解性と具体度である．そこでこの二つを観点にして両仕様記述言語の仕様を比較する．

- 理解性

  まず，プロセス間の通信関係を理解することを考えると，LOTOS仕様はSDL仕様よりも遙かに難

解である．このことは，LOTOSが多重同期通信を採用していることと，インスタンスの生成のしかたで同期関係を決定していることに起因する．また，LOTOSのような多重同期を実世界で実現することは究めて困難なことである．
次に，処理の理解性に関しては，どちらもデータの型定義の理解が大変である．特に，LOTOSでは演算がデータ型の持つオペレーションで記述されているので，その分負荷が大きくなる．

- 具体度
  具体化のための仕様の具体度を比較するならば，SDLではフローチャートレベルの記述が可能であるのに対して，LOTOSではここまでは記述不可であり，具体度は劣りがちである．

以上のように総じてLOTOSの方がSDLより，人による具体化が困難であるが，これは，それぞれの仕様記述言語が対象にしている仕様の違いと，LOTOSが検証のための数学的背景を重視していることから来ている．

(2)機械による具体化

機械による具体化を比較する上では，両仕様記述言語で記述された仕様のスタイルは機械解釈しやすいものであることを前提とする．これは特にLOTOS仕様の記述において言えることであるが，記述スタイルによって機械処理の難易度に大きな差が出ると考えられるためである．この条件を満たす仕様に対して比較しても，先の人による具体化で記述したのと同じことが言え，総じてSDLの方が実現しやすい．しかし，データ型の解釈に関してはどちらも困難であるが，SDLのデータ型の解釈の方がLOTOSの データ型の解釈より困難である．これは，SDLが記述性・理解性のための改良をしており，機械解釈向けでない部分が多いためである．

(3)具体化のためのプログラム言語

それぞれの仕様を具体化する場合のプログラム言語を考えると，次のように整理できる．

- SDLをインプリメントするための言語
  SDLはTaskなどで代入文の記述を基本としている．従って使用するプログラム言語もこのような記述のできるものがよいと思われる．
- LOTOSをインプリメントするための言語
  LOTOSはオペレータの使用法やプロセスの再起生成の部分が究めて関数型言語に近いものが有るので，使用するプログラム言語は，関数型のもののほうが良いであろう．

5. まとめ

本報告では，通信システム用の形式的仕様記述言語であるSDLとLOTOSについて，一般的な比較から，相互解釈，設計と具体化における性質といった多面的な比較・検討を行った．この結果は，設計者がその目的や用途に応じてこれらの仕様記述言語を使い分けるのに参考となるであろう．

謝辞

本検討に際し有益なご助言をいただいた東北大学野口正一教授，白鳥則郎教授に深謝いたします。また，本研究の機会を与えていただいた当社緒方秀夫常務に深謝いたします。

[参考文献]

(1) CCITT : "Functional Specification and Description Language (SDL)", Recommendation Z.100 (1989)

(2) ISO : "Information Processing System - Open Systems Interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour", ISO 8807 (1989)

(3) ISO : " Information Processing System - Open Systems Interconnection - Estelle : A formal description technique based on an extended state transition model", ISO 9074 (1989)

(4) 二木厚吉 : "ISOにおける形式記述技法の標準化動向", 情報処理, 31, 1, pp.3 - 10 (1990)

(5) 丸山勝己 : " CCITTにおける形式記述技法の標準化動向", 情報処理, .31, 1, pp.11 - 19 (1990)

(6) 若原恭 : " SDL言語の特質と処理系の現状と動向 ", 情報処理, 31, 1, pp.23 - 34 (1990)

(7) 高橋薫, 神長裕明, 白鳥則郎 : " LOTOS言語のの特質と処理系の現状と動向 ", 情報処理, 31, 1, pp.35 - 46 (1990)

(8) G. Comparin, G. A. Lanzarone, W. Panzcri and A. Torgano : "Analysis of SDL Specifications Using Petri Nets Techniques", 2nd SDL - User's Forum, PTL (1986)

(9) E. Kettunen and M. Lindgvist : "Toward Practicality of Predicate / Transition Net Reachability Analysis of SDL", the 3rd SDL Forum, SDL'87 : State of the Art and Future Trends, North-Holland, pp.285-294 (1987)

(10) Y. Wakahara and Y. Kakuda : "Verification of Requirements Specification for Telecommunications Software by the Investigation of Specifications-Execution Sequence", Proc. IEEE Globecom 87, Ohmsha, pp.661-666 (1987)

(11) E. Brinksma, G. Scollo and C. Steenbergen : " Lotos Specifications, their Implementations and their Tests", Proc. IFIP WG6. 1, 6th Int. Workshop on Protocol Specification, Testing and Verification , North-Holland, pp.349-360 (1987)

(12) H. Ichikawa, K. Yamanaka and J. Kato : " Incremental specification in LOTOS", Proc. IFIP WG6. 1, 10th Int. Workshop on Protocol Specification, Testing and Verification, North-Holland, pp.185-200 (1990)

(13) C. A. Vissers, G. Scollo and M. V. Sinderen : "Architecture and Specification Style Formal Description of Distributed Systems", Proc. IFIP WG6. 1, 8th Int. Workshop on Protocol Specification, Testing and Verification, North-Holland, pp.189-204 (1988)

(14) S. Nomura, T.Hasegawa and T. Takizuka : " A LOTOS Compiler and Process Synchronization Manager ', Proc. IFIP WG6. 1, 10th Int. Workshop on Protocol Specification, Testing and Verification, North-Holland, pp.169-182 (1990)

(15) K. Yamano, D. Jokanovic, T. Ando, M. Ohta and K. Takahashi : " Formal Specification and Verification of ISDN Services in LOTOS", IEICE Transactions on Communication, E75B, 8 (1992)

(16) T. Ando, M. Ohta and K. Takahashi : " Translation from SDL to LOTOS ", the 5th SDL Forum, SDL'91 : Evolving Methods, North-Holland, pp.95-106 (1991)