マルチメディア通信と分散処理　52－22
（1991．9．26）

仕様記述言語 PAISLey の評価と改良

田島 由樹　　堀川 博史　　佐藤 正則

三 菱 電 機 （株）

機器に組み込むソフトウェアはそのミッションタイムの長さと修正／保守の困難さから、高い信頼性が要求される。これまでの組み込みソフトウェアの開発において発生した不具合は、開発の要求定義の段階で発生したものが多いことがわかった。そこで、信頼性をあげるために組み込みソフトウェアの要求定義段階での検証に米国 AT&T で開発された仕様記述言語 PAISLey を使用し、評価してみた。

評価は、２種類の仕様に対して行ない、複数のプロセスで構成されるソフトウェアを作成する時にその開発の早い段階で使用すると効果的であることがわかった。さらに、評価の作業を分析して PAISLey とその環境の問題点を洗いだした。その問題点を解決するべく、より使い易く、時間制約機能を強化するための改良を行なった。

An Evaluation and Improvement of Executable Specification Language PAISLey

Yuki Tajima　　Hiroshi Horikawa　　Masanori Sato

Mitsubishi Electric Corporation

5–1–1 Ofuna, Kamakura, Kanagawa, 247 Japan

It is necessary to achieve high reliability in embedded computer software, because embedded computer software is long–lived and it is difficult to modify/maintain. We have found that most bugs in embedded software are caused in the requirement phase. We applied specification language(PAISLey by AT&T), to two kinds of software requirement ,and evaluated them. We have found, it is effective to apply PAISLey to requirement specifications of embedded software in systems that consist of some processes, during the early phase of development. Further, we improved the useability of, and the timing constraint facility, in PAISLey.

1 はじめに

機器に組み込むソフトウェアは、ミッションタイムの長さ、修正／保守の困難さから、高い信頼性が要求される。ある機器組み込みソフトウェアの不具合が開発のどの工程で作り込まれるかを分析すると、上流工程で作り込まれる不具合の割合が高い。

そこで、仕様記述言語を用いてプロトタイプを作成し、仕様の妥当性を早期に検証することを試みた。さらに、使用した仕様記述言語の問題点を洗い出し、言語の改良を行なった。

本稿では、仕様記述言語による検証の試行結果と、仕様記述言語の改良について述べる。

2 仕様記述言語 PAISLey

仕様記述言語としては、米国 AT&T で開発された PAISLey(Process-oriented Applecative Interpretable Specification Language) を採択した。PAISLey を選択した理由は、PAISLey が組み込みシステムを対象とした言語であることである。PAISLey には、その他に次にあげるような特徴がある。

- 周期的な処理を基本としていること
- 関数型言語であること
- 並列動作を記述できること
- 未定義部分を含むものを処理できること
- 時間制約を記述できること

二つの入力に対する論理積をとる仕様の PAISLey の記述例と実行結果例をそれぞれ図 1と図 2に示す。

```
(operate[False]); ‘‘ プロセスの定義 ’’
operate : BOOLEAN --> BOOLEAN;
   ‘‘operate の入出力の型宣言 ’’
operate[state] = and[input];
   ‘‘operate の定義、 input の出力に対して
     論理積をとる ’’
input : --> BOOLEAN * BOOLEAN;
    ‘‘input の入出力の型宣言 ’’
```

図 1: PAISLey の記述例

仕様記述は、設計に入る前の要求定義に対して行なった。従来は、人間が自然言語で書かれた要求定義をレビューすることによって上流工程における信頼性を上げていた。これに対して、PAISLey を用いると仕様を機械処理可能な言語で記述し、シミュレーションすることができる。

```
  時間       関数の初期化／終了
0.00000 operate[False]  ‘‘operate の初期化 ’’
0.00000 input[]         ‘‘input の初期化 ’’
0.06525 input=(True,True)   ‘‘input の終了 ’’
0.69481 operate=True       ‘‘operate の終了 ’’
0.69481 operate[True]
0.69481 input[]
1.10366 input(False,False)
1.34287 operate=False
```

図 2: PAISLey の実行結果例

3 記述評価

今回の試行の目的は、PAISLey を組み込みソフトウェアに適用する際の、言語の習得容易性、記述性、適用可能性を評価することである。そのために、記述の対象として性質の異なる 2 種類のモデルを選択し、要求定義の詳細さのレベルを変えて記述してみた。さらに、PAISLey の利点／欠点を明確にするために他の言語で同一のモデルを記述してみた。

対象として選択した 2 種類のモデルを次に示す。このモデルはすでに開発が完了している実際のソフトウェアの要求定義から抽出したものである。

モデル 1　処理機能は複雑であるが 1 プロセスで表される。

モデル 2　処理機能は単純であるが複数プロセスで表される。

モデル 1 については、基本仕様と概略仕様の 2 種類の仕様を記述した。概略仕様は基本仕様よりも詳細な仕様である。

3.1 モデル 1 − 1 プロセス −

モデル 1 の要求定義は、自然言語により書かれている。

最初に、基本仕様に対して従来のレビューによる方法と PAISLey を用いた方法の両方を用いて要求定義段階での検証を行なった。その結果、それぞれの方法によって発見された問題点の数、不具合と認定されたもの（仕様に反映されたもの）の数、及び作業時間を表 1に示す。PAISLry の記述は、レビュー後の要求定義仕様に対して行なったので、発見された 6 件の不具合はレビューでは発見できなかったものである。PAISLey で発見できた 6 件の不具合が総合試験規格書のレビュー時に発見されたとすれば、その改修作業には 1 5 6 時間が費やされる（仕様変更の影響が 1 モジュールに収まったと仮定している）。従って、7 4 時間の作業時間削減となる。

次に、詳細さの異なる概略仕様に対して PAISLey 記述を行ない、その結果と基本仕様の結果を比較した（表

表 1: レビューと PAISLey の比較

	レビュー	PAISLey
指摘された問題点	9件	20件
検出された不具合	4件	6件
作業時間	4時間	82時間

2 参照）。概略仕様の記述は不具合の発生しそうな部分に対して行なった。

表 2: 詳細さの異なる記述の比較

	基本仕様	概略仕様
検出された不具合	4件	6件
作業時間	37時間	63時間

不具合1件を発見するのに費やした作業時間は、基本仕様書も概略仕様書も約9時間であり差がない。

3.2 モデル2 －複数プロセス－

モデル2の要求定義は、プロセス間の関係を示すデータフロー図とタイミングチャートを用いた動作例記述からなる。今回の記述対象としては、ハードウェアで実現される部分も含めている。

PAISLey の作業に関するデータは表3の通りである。不具合の多くは、記述不足、記述抜けにより、複数の解釈が成立することによるものが多い。検出できた不具合数がモデル1に比べて多いのは、要求定義の厳密さのレベルが幾分低いことによる。

表 3: データ入出力に対する作業データ

検出できた不具合数	15件
作業時間	102時間
予想修正時間	390時間
作業時間の削減	288時間

3.3 C言語との比較

他の言語との比較のために、同一の仕様を PAISLey と C言語で記述した。その結果を比較したものを次に示す。

1. モデル1の場合

モデル1に対する PAISLey と C の比較を表4に示す。

記述ステップ数は同程度である。作業時間では C言語の方が有利である。作業時間をさらに細かく調べると、開発時間は PAISLey も C言語も同程度であるが、言語習得時間は PAISLey は C言語の3倍近くかかっている。

以上をまとめると PAISLey は C言語と比較して

表 4: PAISLey と C の比較（モデル1：基本）

	PAISLey	C
ソース行数	101行	115行
作業時間 (習得時間)	23時間 (17時間)	11時間 (6時間)

優位ではなく、習得容易性ではむしろ劣っている。

この結果は、より詳細な記述を行なうとより顕著になる。基本仕様をより詳細にした概略仕様に対する比較を表5に示す。

表 5: PAISLey と C の比較（モデル1：概略）

	PAISLey	C
ソース行数	308行	219行
作業時間	13時間	7時間

2. モデル2の場合

モデル2に対する PAISLey と C の比較を表6に示す。

表 6: PAISLey と C の比較（モデル2）

	PAISLey	C
ソース行数	125行	338行
作業時間	8時間	14時間

複数のプロセスでデータ交信を行なう機能が重要となるものについては、C言語で記述することは可能であるが、開発時間も記述量も PAISLey の方が少ない。

3.4 評価のまとめ

3.1と 3.2の結果をまとめると、仕様記述による検証によってモデル1では6件の不具合を、モデル2では15件の不具合を発見することができた。そして、それらの不具合が要求定義の段階で発見されず、総合試験規格書のレビュー時に発見されたとするときに、フィードバックのためにそれぞれ156時間と390時間がかかることになる。今回、PAISLey での記述・実行にモデル1の場合で26時間、モデル2の場合で102時間かかっている。PAISLey を使用しなかった場合にフィードバックにかかる時間との差をとると PAISLey を使用したことによって削減された作業時間が求められる（表7 参照）。従って、要求仕様分析・定義フェーズへのプロトタイピングの適用は、仕様自体の妥当性を早期に検査し、品質を高めることに効果がある。

また、PAISLey の使用については次のことがいえる。

表 7: 仕様レベルの検証によって削減される時間

	モデル 1	モデル 2
記述時間	8 4 時間	1 0 2 時間
フィードバックに要する時間	1 5 6 時間	3 9 0 時間
削減時間	7 4 時間	2 8 8 時間

- モデル 1 で基本仕様と概略仕様で不具合発見にかかる時間に差がない
- C 言語との比較から、モデル 1 (単一プロセス) では、基本仕様について PAISLey 記述の有意性が見られない。概略仕様については、C 言語の方が有意である。
- モデル 2 (複数プロセス) では PAISLey 記述の有意性が見られる
- PAISLey は手続き的な処理の記述には向いていない。

これらのことから、PAISLey は対象とするソフトウェアの性質と適用段階により、得られる効果に差があることがわかった（表 8 参照）。

表 8: PAISLey の有効性

	開発のはじめに 小さなものを作る	開発の途中に ある程度の規模を作る
複数プロセス	○	△
単一プロセス	△	×

PAISLey は、複数プロセスで構成されるソフトウェアを作る際に、その開発の早い段階で、概略のアルゴリズムが正しく作動するかを調べる場合に、使用することが推奨される。小さな規模でも複数プロセスを扱うとなるとタイミング等が関係して複雑になることが多い。そのような複雑なアルゴリズムを早い段階で検証することは、開発の効率化につながる。

C 言語との比較の結果、PAISLey 言語の習得にはかなりの時間がかかることがわかった。特に手続き的な処理の記述に向いていないため、複雑な処理を記述できるようになるためには、時間がかかる。

4 PAISLey の改良

記述評価の結果、PAISLey は適用対象と適用フェーズを限ると開発の効率化に有効であることがわかった。そこで、記述評価の作業を分析して洗い出した PAISLey とその環境の問題点を解決するために、使い勝手と時間制約機能に関する改良を行なった。

4.1 改良項目の決定

改良項目決定のための手順を次に述べる (図 1参照)。改良項目導出には、適用検討のために行なった記述作業の記録を利用している。

1. 問題点を洗い出す
   PAISLey の適用検討の記述作業記録を分析することによって、PAISLey とその環境についての問題点を洗い出す。
2. 改良項目を求める
   ブレーンストーミングにより、1で洗い出された問題点をその問題を解決するために改良すべき PAISLey の機能に置き換えて改良項目とする。
3. PAISLey 作業モデルを作成する
   適用検討の記録と仕様記述量から、PAISLey を使用して行なう検証作業の標準的作業モデルを作り出す。
4. 実際に改良する項目を決める
   1) 改良項目毎にその項目を改良した場合に削減される作業時間を求める。作業時間の見積もりに 3で求めた作業時間モデルを使用する。
   2) 改良項目毎に改良に要するステップ数を求める。
   3) 改良に要するステップに対する削減される作業時間の割合を求める。
   4) 全作業で求めた割合で改良すべき項目を設定する。

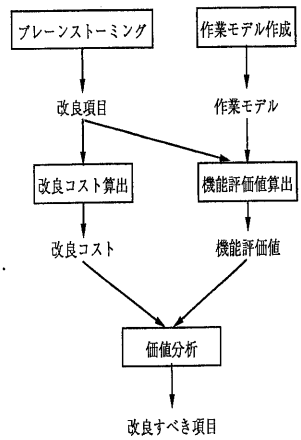

図 3: 改良項目導出手順

2のブレーンストーミングは、仕様記述者、作業分析者、改良作業者の4名で行なった。改良項目として42項目を導出した。3で求めた作業モデルは、1067ステップ100写像を57.7時間で記述するものである。各作業時間は、仕様理解・設計に11.3時間、コーディングに23.2時間、デバッグに8.5時間、テストに14.7時間である。4-1)の削減される作業時間の見積もりは、実際に仕様を記述した2名で行なった。4-4)では、4-2)で求めた割合が高いものから順に次の6項目を改良項目に決定した。

1. 仮パラメータの外側の括弧の省略
2. 時間比較の精度をあげる
3. 関数表現をパイプ表現に変える
4. トレースファイルを常に新しくする
5. 中置記法の導入
6. 関数の初期化の定義を変える

1から7までは、記述し易さ、読み易さなどの使い勝手に関する改良であり、8)はPAISLeyの時間制約の機能を強化する改良である。

4.2 改良内容

4.1章で導出された改良項目のうち、使い勝手に関する改良3)と時間制約の機能に関する改良6)について述べる。改良作業に入る前にPAISLeyインタプリタの解読を行なった。

1. 仮パラメータの外側の括弧の省略

   関数に組形式の引数を与える場合に次のように記述する。

   ```
   and[(False,(False,True))];
   ```

   引数の括弧[ ]の中にさらに組形式であることを示す括弧( )を書かなくてはならない煩わしさがある。この煩わしさを解決するために、引数に与える仮パラメータの一番外側の括弧( )の省略を可能にした。改良後の入力は次のようになる。

   ```
   and[False,(False,True)];
   ```

   PAISLeyの処理系は、括弧( )で括られたものを組形式と認識する。そこで、PAISLeyの処理系に渡す前に引数の括弧[ ]の内側に必ず組形式の括弧( )を挿入するようにして実現した。

2. 関数表現をパイプ表現に変える　PAISLeyは関数型の言語なので、手続き的な処理を記述するためには関数をネストして呼び出すことになる。関数のネストは次のように記述される。

   ```
   a[start] = b[ c[ d[start] ] ]
   ```

   この表現には次のような欠点がある。

   - 引数の括弧[ ]がネストして見にくいものになる。
   - 記述する括弧[ ]の数が多くなる。
   - ネストした関数の中側のから処理が行なわれる。
   - PAISLeyでは、時間的な制約条件を関数の評価時間を指定することによって記述する。上の記述では関数dの処理時間は関cの処理時間に含まれない。

   記述のし易さ、読み易さ、処理に対する自然さを増すために、関数のパイプ表現も行なえるようにした。改良後、先の例は次のように記述される。

   ```
   a[start] = d[start] | c[ ] | b[ ] ;
   ```

3. トレースファイルを常に新しくする

   *1em PAISLeyはインタプリタで実行することが可能である。インタプリタの実行結果をトレースファイルに保存することができる。トレースファイルの名前は、ユーザがインタプリタの中で指定する。そのトレースファイルがあれば、出力がファイルに追加され、なければ新たに作成される。

   記述評価を行なった時に、インタプリタで実行途中で設定を変えた場合にトレースファイルを消すのが面倒であるとの声が聞かれた。そこで、トレースファイルを常に新しく作成することにした。

   従来はトレースファイルがアペンドモードでオープンされていたものをオーバーライトするように変更した。

4. 中置記法の導入

   PAISLeyは関数型の言語であるために、演算子の中置記法は許されていない。すべての演算を関数呼び出しによって行なう。例えば比較演算は次のように記述する。

   ```
   equal [ (True, False) ]
   ```

   従って、記述量が増加し、同時に記述した仕様の読みやすさも損なうことになる。そこで、演算子の中置記法を導入することにした。今回は、特に利用頻度の高い、比較演算子=の使用を実現した。改良後は次のような記述になる。

   ```
   True=False
   ```

5. 関数の初期化の定義を変える

関数の中でプロセス間通信を行なう交換関数を使用する場合に、時間制約条件が守られない場合があった。これを直すために関数の初期化の定義を変更すれば良い。

時間制約条件の守られない場合について2つのプロセスが交信を行なう簡単な例で説明する。図4の

```
(add-0[0],add-1[0]);

add-1:  INTEGER --> INTEGER;
add-1[start] = sum[(x-sensa[start], 1)];
add-1: ! lb 10.0ms, ub 10.0ms;

add-0: INTEGER --> INTEGER;
add-0[a-flag] = sum[(xr-sensa[a-flag],0)];
add-0: ! lb 5.0ms, ub 5.0ms;
```

図 4: PAISLey 記述例

PAISLey の記述例は、add-0 と add-1 の2プロセスがそれぞれ1と0を足し込む計算を行ない、その結果を x と xr の関数で交信を行なうものである。

この記述の実行結果は図5に示す。図5の process-1 の xr-sensa の評価は (1) で始まり、(3) で終了している。その間に process-2 の x-sensa の評価が (2) で始まっている。従って、(1) の xr-sensa と (2) の x-sensa は交信を行なうはずであるが、実際には (2) の x-sensa は (4) の xr-sensa と交信を行っている。そのために本来なら指定時間内に評価が終る関数 add-1 の評価時間がオーバーして (6) の時間制約エラーが出る。これは、インタプリタの処理が、xr が先に

```
 process-1   process-2
25.00000 xr-sensa[5]                (1)
25.41767             x-sansa[4]     (2)
29.45955 xr-sansa=5                 (3)
30.00000 add-0=6
30.00000 add-0[6]
30.08370 xr-sensa[6]                (4)
30.08370             x-sansa=6      (5)
30.08370             add-1=8
30.08370             add-1 10.08370 ms !!!!(6)
```

図 5: 改良前の実行結果

評価された場合に交換しないようになっているためである。そこで、インタプリタの処理を xr が先に評価された場合には交換を行なうように変更した。改良後の PAISLey の実行結果を次に示す。

```
          process-1    process-2
10.00000 xr-sensa[2]
10.00000              add-1[2]
10.00000              x-sansa[2]
13.61082 xr-sensa=2
17.99655              x-sansa=2
```

図 6: 改良後の実行結果

5 おわりに

仕様記述言語 PAISLey を機器組み込み用ソフトウェアに対して適用することを検討してきた。今回、2種類の実際に使用された要求定義を用いて、PAISLey の記述評価を行なった。その結果、PAISLey は、適用する仕様の性質や、適用する開発フェーズによって効果に差があることがわかった。PAISLey を効果的に使用するためには、複数のプロセスで表現される仕様に対して、設計の早い段階で適用することが奨められる。

記述評価の過程から PAISLey とその環境に関する問題点が導出された。それらの問題点のうち使い勝手に関するものと時間制約機能によるものについての改良を行なった。改良を行なったことにより、記述しやすさ、記述の読みやすさが増し、複数プロセスの間で交信を行なう仕様の時間的な制約条件をより正確に検証できるようになった。

参考文献

[1] P.Zave:An Operational Approach to Requirements Specification for Embedded System,IEEE SE ,1982.3.

[2] P.Zave:An Operational Approach to Requirements Specification for Embedded System.IEEE SE,1982.3.

[3] P.Zave:PAISLey REFERENCE MANUAL,AT&T, 1988.

[4] 堀川 他: 仕様記述言語 PAISLey の適用検討, 情処全大 41-1G-4,1990.

[5] 菅野他：日科技連ソフトウェア品質管理シリーズ6：ソフトウェアの生産技法