脳に学ぶ超可用性ソフトウェアの実現に向けて
Toward a High-availability Software Architecture inspired by Brain

山川宏[*1]
Hiroshi Yamakawa

[*1] (株)富士通研究所
FUJITSU LABORATORIES LTD.

Avoiding an explosive increase of the cost of operation and maintenance in software is structurally difficult. To overcome this problem, I want to design high availability software architecture inspired from non-stop software principle of a brain. We consider that super-substitutability nature of execution program code should be imitated, which we think a source of high availability of the brain. Program code volume becomes large with super-substitutability, and code modification by human becomes difficult. In this paper, a hybrid software architecture is proposed. In this architecture the code developed by a usual computer language is converted into the code set with super-substitutability, and it is executed with high availability.

1. はじめに

システムの信頼性を表す代表的な指標としては，ハードウエアやソフトウェアが故障する頻度や期間が短い事を示す信頼性，ユーザから見てシステムをできるだけ長時間使用可能な状態に保つ事を示す可用性(Availability)，保守し易い事を示す保守性がある．一般に信頼性が低いと可用性が低くなるが，障害の影響の局所化などにより，可用性への影響を小さくでき，保守性が低いと障害発生時の修理時間が長引くため可用性が低くなる．

将来，IT が社会インフラとして一層世の中に浸透するならば，そこで用いられるソフトウェアは，修正を加えられながらも動き続ける高度な可用性が要求されるであろう．しかし近年ソフトウェア市場の現状を見ると，運用・保守コストが増大し，実にIT投資の約 7 割が運用・保守に費やされている． こうした状況の原因としては，一つ目には「システムが'つぎはぎ'で運用が複雑になる」，二つ目には「機能追加／修正などの柔軟性が低い」などがある(ガートナー・ジャパン:2006)．つまり現状においてソフトウェアの可用性が高く保たれているとは言い難い．

Gail Murphy 氏は，プログラムの保守に際しては，しばしば想定とは異なる設計のプログラムコードに直面し，膨大なソースコードの森の中で迷ってしまうと，指摘している[山本 08]．そのため全体像を把握してした改修は困難で，'つぎはぎ'的な対応となりがちで，コードの保守性をさらに低下させてしまう．今後，ソフトウェアがさらに大規模化・複雑化しつつ過去の遺産を引き継いでゆけば，運用・保守コストはさらに増加し，近い将来において，ソフトウェア投資のほとんどが運用・保守費に占められる「運用保守爆発」といった事態が起こる危険性も想定できる．

こうした課題に対してソフトウェア工学では，例えばソフトウェアメトリクスを用いて保守を効率化する手法や[山浦 05]，コードクローンに着目したソフトウェア保守支援ツール[山科 08]などといった様々な研究・開発が進んでいる．しかし逆に，運用開発者に提供される情報が多すぎることが問題となり始めており，運用開発者の行動とツールの提供する情報との間の不一致を解消することが支援を向上させるとの指摘もある[山本 08]．何れにしてもソフトウェア工学の主流となるアプローチは，人によるソフトウェア・ハンドリング能力の拡大であり，そのための様々な支援ツールの提供をおこなっている．

連絡先：山川宏，(株)富士通研究所 ソフトウェア＆ソリューション研究所 ソリューションテクノロジ研究部， 川崎市中原区上小田中 4-1-1，044-777-1111， *ymkw@jp.fujitsu.com*

一方で，こうした運用保守爆発には構造的な背景があり，それ自体を避けることは難しい．何故なら計算機の誕生以来，その処理能力やメモリの指数関数的な増大は継続しており，ソフトウェア資産も線型以上に増大している．このため計算機リソースを運用保守するためのコストも爆発的に増加している．

今後，ソフトウェアア学により人によるソフトウェア運用保守能力をさらに向上させたとしても，将来的には保守対象となる計算機リソースの爆破的増大には追随しきれない可能性が高い．よって運用保守爆発を克服するにはソフトウェアの運用保守において自動化できる部分を大幅に増やしつつ，可用性を高める技術が必要であろう．

2. ソフトウェアの可用性を脳に学ぶ意義

現存する知的システムにおいて，発達や学習により機能を変化させつつ，ノンストップで動き続けるソフトウェア備えているのは，生物の脳をおいて他には無い．さらに脳における機能追加が，通常のソフトウェア工学ではネガティブに扱われる，'つぎはぎ'的に行われる点も興味深い．そこで脳を参考にして可用性の高いソフトウェアアーキテクチャの構築を目指す検討を行う．

これまでにもニューラル・コンピュテーション(NC)ブームは二度あった．そうであるにも関わらず，再度，脳に学んでソフトウェアアーキテクチャを研究する意義としては以下がある．

(1)実行コード不可読性の容認： 実行コードに対して NC 等を含む機械学習を利用して修正すれば，人が指示すべき修正情報を大幅に減らせ得る．しかしそうした実行コードは人にとっては可読性が悪化しやすく，一般に実システム適用では敬遠され勝ちである．一方，現状既に，大規模レガシーソフト等の人によるコード理解は困難となりつつあるので運用保守する実行コードに対しては人の理解によるコードの直接編集を諦めれば，可読性を悪化させるデメリットは容認しうる．**(2)「可用性」に着目したNC**: 従来の NC ブームでは，神経素子のネットワークから出発した分類器としての工学応用や，高次の思考能力の理解などの研究が主流であり，「学習・適応しながらの高可用性の実現」に目標設定した先行研究は見あたらない．必ずしも脳の全容の理解に至らずとも，目標を特定化することで工学的に意義ある成果を得られる可能性が高まる．**(3)脳科学等の進歩による知見の蓄積**: 近年の脳科学や分子生物学などの長足の進歩により，脳機能に関わる知見が急速に蓄積されつつあり，20 年前では不可能であった実験により新たな知見を得られる．**(4)安価で膨大な計算リソース**: 処理能力・メモリなどの計算リソースが安価に大量に利用できるようになり，システム保守等の，本来実行すべき処理以外に計算リソースを配分することが日常化している．

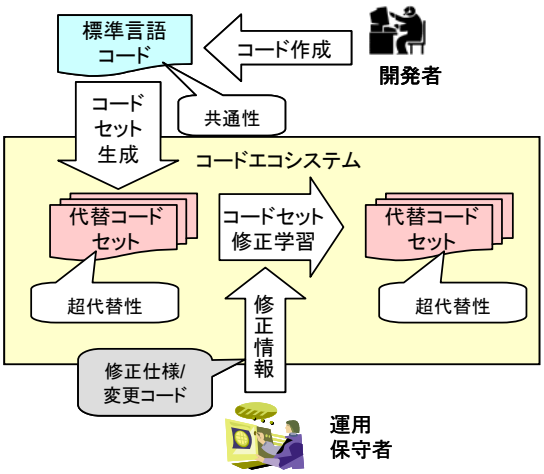

図 1: 混合ソフトウェアアーキテクチャ

3. ソフトウェアの可用性と超代替性

システムの可用性はアーキテクチャ・レベルにおいて，どのように達成されるのだろうか．ハードウエアの可用性を高めるには同一機能を持つ物理的なリソースを冗長化すれば障害発生時等にリソースを切り替えて動き続けることができる．これに対し，ソフトウェアでは，全く同一機の能を複数個用意しても，同じ障害を発生させることが多いため可用性向上には結びつきづらい．

人の脳が一生を通して動き続けられる能力のソフトウエア的な側面は，課題の目的を頭にとどめながら，その目的達成のために柔軟に動作を変更して試行錯誤を繰り返す(セットの転換)能力との関係が深いと思われる．こうした脳の能力の背景には，脳の処理系が，「超代替性＝特定課題に対し，柔軟で多様な代替解決手段を有する」という性質を備えているからであろう．そこで超代替性を一般的なソフトウェアに巧く取り込むことができれば，例えば動作環境と仕様との齟齬や不完全なコード修正によって生ずる障害やデッドロック等を柔軟に回避できる能力を実現することで，可用性を高めうると考えられる．

もちろん現状のソフトウェアでも，予め複数の処理モジュールを用意すれば，それらを切り替えて実行できる．しかし，人手により代替手段を用意する場合は，利用可能性が低いコードを含むほどに多様性を高めることは通常行わない．また連想メモリ的な機構をベースとする脳の情報処理では自在に('場当たり的'かつ'つぎはぎ的'に)代替手段を追加することを許容できているが，標準的なソフトウエアでは，開発者等が扱い易くなるように代替する処理モジュールのインタフェースを揃えることになり，代替手段の柔軟性に制限が加わりがちである．

そこで，脳のソフトウェアとしての超代替性の原理を参考にし，可用性の高い新しいソフトウェアアーキテクチャを実現する道筋について検討をおこなう．

4.混合ソフトウェアアーキテクチャ

実行コードに超代替性を取り込むことで可用性の高めるという方針は，ソフトウェア工学においてコードの共通性を高めようとする方針とは明らかに相反する．一般的には，ソフトウェアの設計／開発およびソフトウェアアーキテクチャ全体は手作業であり，コードの共通化を進めることで，全体の見通しを良くし，バグフィックスや仕様変更の際に書き換えるべきコードを減少させて保守コストを低減し，書き換え忘れ等に起因するバグ混入等による信頼性低下リスクを減らしている．

しかし，コードに超代替性を導入すると，その記述量が多くなり人手による保守には不適となる．そこでコードの保守では運用保守者が与えた少ない情報の意図を汲んでシステムがコードを自動的に修正学習できるような仕組みを開発することで，保守コストを低減しつつ可用性を高めたい．しかし機械で修正されたコードは一般に人にとって読み辛くなりコード可読性はますます低下し，コードを直接理解することは困難になる問題がある．だが，実行コードを間接的に低コストで保守できるなら，2 節で述べたように実行コードの不可読性は許容しうるだろう．

運用保守段階における実行コードが可読でなくても，設計/開発時には人にとって理解しやすい標準的な計算機言語を使うべきであろう．そこで，本稿では開発し易さと実行時の可用性を両立するため，図 1 に示すように，開発時には標準的計算機言語でコーディングを用い，それに対し何らかの変換を加えることで生成される超代替性をもつコードの集合体(これを代替コードセットと呼ぶ)と，これを実行する処理系をもつ「混合ソフトウェアアーキテクチャ」を提案する．なお代替コードセットおよびその生成・実行・修正学習する環境を併せコードエコシステムと呼ぶ．

混合ソフトウェアアーキテクチャの主要な処理としては，まず開発者が可読性の高い標準言語を用いてコードの設計／開発を行う．次にこれを何らかの手段で代替コードセットにコード変換する．コードエコシステムによる実行では多様な実行コードを柔軟に選択して利用することで可用性を高める．そしてコードエコシステムにおけるコード保守では運用保守員が準備した修正情報を利用して代替コードセット対しコード修正学習を行う．

修正情報は人手で作製するため，コストの面から超代替性を持たせられず，修正学習のプロセスを工夫して代替コードセットの超代替性を維持せざるを得ない．おそらくここでは脳において海馬に蓄えたエピソディックな短期記憶を，大脳を中心とした長期的な記憶にトランスファーする仕組みが参考となるだろう．

5. まとめ

ソフトウェアにおける運用保守爆発が構造的に回避困難であり，その克服には可用性の高い脳の基本原理に学ぶことが有力なアプローチであることを指摘した．脳機能の全容の理解に至るまでの道のりは遠いが，「可用性」という特定機能に着目して脳から学ぶことで工学的な成果を得たい．そうした点で脳全体のシミュレーションを目指す IBM の[Blue Brain Project]等とは異なる．

本稿では，脳における高可用性は，ソフトウェアの実行コードに超代替性をもつことで実現していると仮定した．そうした代替コードセットの記述量は多量となるために，その保守にはシステムによる自動的なコード修正が必要となり，それ故にコード可読性を犠牲にせざるを得ない．そこで開発時には通常の計算機言語を用い，それを変換した超代替性をもつコードセットにより実行する混合ソフトウェアアーキテクチャを提案した．

なお本研究は 2008 年秋に開催された(株)富士通研究所創立 40 周年記念イベント「夢コンテスト」にて，優秀賞を獲得した提案「脳型超可用ソフトウェア」を起点とし実施中である．しかし，未だ構想レベルであり実現に向けては課題が多く，今後は，関連諸分野の研究者との連携を含めた研究開発を進めたい．

参考文献

[山本 08], 山本里枝子他,ソフトウェアエンジニアリングシンポジウム 2008 開催報告，情処理学会研究報告，2008-SE-162, pp.9-16, 2008.

[山浦 05] 山浦恒央，初めて学ぶソフトウェアメトリクス，日経 BP 社，2005.

[山科 08]山科隆伸他，コードクローンに着目したソフトウェア保守支援ツールの設計と実装，信学技報，vol.108, no.64, pp.65-70, 2008.

[Blue Brain Project] http://bluebrain.epfl.ch/