Vol.2015-SE-187 No.39
2015/3/13

ネットワークリプレイツール PARROT による テスト自動化のケーススタディ

森拓郎[†1]　秦野康生[†1]　臼井崇文[†1]　田中修一[†1]

概要: 計算機システムの高機能化に伴い，機能テスト工程は複雑かつ高コストとなる傾向がある．近年，ウェブシステムなどの標準的なシステムを中心にテスト自動化向けツールが拡充されている一方で，事業領域独自のアプリケーションを含む非標準的なシステムでは固有の要件への対応の困難さから自動化が遅れている．そこで，システムの要件による影響が少ないネットワーク通信に着目し，計算機間での通信処理をリプレイしテストを自動化するツール PARROT を開発した．本稿では，本ツールをエンタープライズシステムに適用したケーススタディを報告する．

Case study of Test Automation using PARROT Network Replay Tool

TAKURO MORI[†1] YASUO HATANO[†1]
TAKAFUMI USUI[†1] SHUICHI TANAKA[†1]

abstract: Functional testing has tendency to increase of cost with high-functioning of computer system. Recently, test automation tools are mainly focused on commodity computer systems (e.g. web system) and improved, whereas domain specific computer systems are behind because of the difficulty of supporting unique system requirements. Therefore, we focused on network communication what is less impact by system requirements, and developed PARROT software tool to test automation by network message replay. Here we show that case study of applying PARROT with an enterprise system.

1. はじめに

情報システムの構築では，テストに費やすコストが製品の初期開発コストの 45%を占める[1]ため，コスト削減のためにはテストコストの削減が必須課題である．特に複数の計算機ノードが連携するサブシステム間のシステムテストでは，動作が複雑になりやすいためテスト件数の増大や問題解析の困難化，修正後の確認における再現性の低下などコスト増加の要因が発生しやすい．

そこで，テストの実施および結果確認の自動化が検討されるが，既存システムに対して後付けでテスト自動化を導入する際には自動化範囲が限定され，費用対効果が低くなる傾向にある．

本稿では，サブシステム間のテストにおいて，通信履歴情報であるパケット情報を活用したメッセージリプレイ技術によるテスト自動化に着目し，メッセージリプレイソフトウェア PARROT（Packet Analysis and Respondent Replay Operation Toolset）を開発した．設計にソフトウェアアーキテクチャ技術を取り入れることで，実際のシステムへの適用の際には導入コストを抑え，300 万件のテストを自動実行してテスト工程のコスト削減を実現した．本稿ではその結果を報告する．

2. テスト自動化とその課題

本章では，テスト自動化ニーズの高いデータバリエーションテストを例に，テスト自動化の課題を示す．

データバリエーションテストは，テスト対象に対して仕様から想定される様々なデータを入力して動作や応答を確認するテストである．正常なデータだけでなく異常なデータ等も入力してシステムの挙動をテストできる一方で，テスト対象となる仕様の範囲が広がるほどテストの項目は増加する．さらに，システム改修に伴う回帰テストでは同様のデータバリエーションテストを繰り返す場合があり，さらなるコスト増要因となる．そこで，人手で行っているテスト作業の自動化によるコスト削減が検討される．

テスト自動化において重要なのは費用対効果である．データバリエーションテストは類似のテストの繰り返しが多いため効果は大きいと考えられやすいが，以下に示す自動化のためのコストの増加要因の課題を解決しなくては得られる効果は限定的である．

テストケース作成のコスト

テスト自動化コストはスクリプト等の開発とそのメンテナンスで構成され，テスト対象の動作が複雑になるほど増大する．

周辺装置の準備

テスト実施環境を構成する機器はテスト対象のみならずテスト対象と連携する周辺の装置を含むため，テスト環境ごとに周辺装置の入手や準備が必要となる．

問題の再現性

テストにより問題が発覚した際には，現象の再確認や問題解決後の再試験において問題発覚時と同様の入力を再現する必要がある．再現性が低い場合には繰り返しテストを実施することとなり煩雑である．

†1 (株)日立製作所
Hitachi Ltd.

3. 従来技術

テスト自動化のために用いられてきた手法として，簡易かつ効果が得られやすいのは軽量プログラミング言語を用いたスクリプトによる自動化である．しかし，スクリプトは簡易にテストを自動化できる一方で，要するコストはスクリプトの設計や開発のみならず運用や管理を含み，適切な管理を欠いた際にはテストの品質の低下ひいてはコスト削減の失敗に繋がる[2]ことが知られている．

そこで，実際のシステムの処理内容を用いて，その処理を再現することでテスト（リプレイテスト）を行うキャプチャ・リプレイ技術が開発されている．この技術では，実際の処理内容からテスト自動実行の入力情報を作成するため，スクリプト作成作業が低減され，バリエーションの作成作業は入力情報をベースの改変および追加によりゼロベースでの作成と比較して簡便となる．また，サーバ装置のリプレイはサービス仮想化とも呼ばれ，周辺装置を模擬するため機器および準備双方においてコスト削減が期待できる．

特に，図 1 に示すような，パケット情報を利用した計算機ノード単位でのキャプチャ・リプレイ（メッセージキャプチャ・リプレイ）はコスト削減効果の高さから盛んに技術開発が行われいている．例として，CA 社はウェブアクセスのパケット情報からサーバおよびクライアントの挙動を再現するツール[3]を展開しており，Oracle 社は DB アクセスのワークロード情報からトランザクションを再現する製品[4]を提供している．

既存のメッセージキャプチャ・リプレイ技術は標準的な通信プロトコルやサービスに特化したものが多く，それらについては効果が高い一方で，独自のプロトコルを持つシステムや，複雑な処理ルーチンが追加されている場合おいては拡張性に課題が残る．

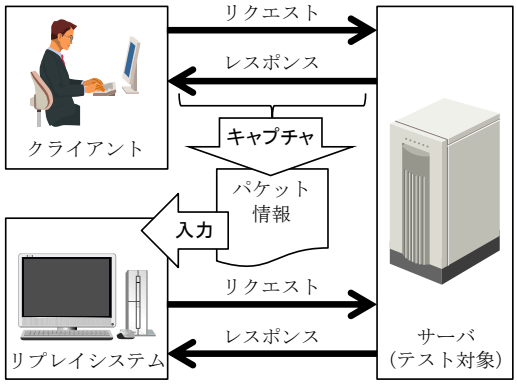

図 1　パケットキャプチャ・リプレイの例

4. 解決方法

本稿では，独自プロトコルや独自ルーチンを持つシステムについて対応が容易なメッセージキャプチャ・リプレイ技術の開発とリプレイシステムの再生により，それらシステムに対する自動テストを実現する．以下ではそのための解決方法について述べる．

(1) メッセージ情報

リプレイを行う際に入力となる通信内容に関する情報をメッセージ情報と定義する．アプリケーションが 1 つのメッセージを送信した際，その情報は複数のパケットに分割されることが一般的であり，プロトコル対応のためには複数のパケットからアプリケーション単位の情報を構築する仕組みを用意する．さらにメッセージ情報はデータバリエーションテストでの適用を考慮し編集を容易とする．

(2) スケジュール情報

リプレイを行う際の挙動を定義する情報をスケジュール情報と定義する．Fowler の定義[5]に照らせばリプレイシステムはネットワークにおけるモックであると言えるため，スケジュール情報には，リプレイシステムがリプレイを行う契機およびリプレイ内容が定義される．またメッセージ情報と同様にリプレイの柔軟性を向上させるための編集の容易性も備える．

(3) プロトコル拡張の容易化

リプレイシステムは IP のようなネットワーク層と，TCP や UDP のよなトランスポート層のプロトコルに加え，上位アプリケーションのプロトコルの内容を解析し，リプレイの際には必要に応じた内容の編集を行う機能が必要である．例としては HTTP におけるタイムスタンプが挙げられる．

そのためリプレイシステムではプロトコルごとに処理ルーチンを持つ必要があるが，既存プロトコルのルーチンを流用しつつ新規に必要なプロトコルのみ実装することで極力この実装工数が増大しないようなアーキテクチャとする．

(4) 拡張機能

リプレイシステムのユーザ独自の拡張によるリプレイ内容編集機能を備える．

5. 用語の定義と準備

本稿で用いる用語と，リプレイテスト実施のためのパケット情報を収集する手法について述べる．

PARROT

本研究により開発したリプレイツールの名称であり，メッセージ情報の解析およびリプレイを行う．

パケット情報

ネットワークを介して計算機ノード間でやり取りされる通信情報の単位である．Ethernet 環境であれば Ethernet フレームを指す．

メッセージ情報

ネットワークを介してアプリケーション間でやり取りされる情報の単位であり，本稿でメッセージリプレイを行う際の実行単位でもある．1 個以上のパケット情報により構成される．

リプレイ情報

PARROT がリプレイを行う際に必要な情報すなわちメッセージ情報とスケジュール情報である．

5.1 パケット情報の収集

リプレイに先立ちパケット情報を収集する．パケット情報の収集方法は，一般的なツールである tcpdump[6]や Wireshark[7]が利用可能であり，その手法は図 2 に示す 3 種類に分類できる．各手法メリット・デメリットが存在するためシステム構成と要件を鑑みて検討すべきである．

(1) 個別キャプチャ

クライアントやサーバなどの計算機ノード上で行うパケットキャプチャによって，そのノードが送受信したパケットの情報を得られるが，計算機ノードの性能によっては，キャプチャするパケット量の増大に伴いパケットロスが発生する．また計算機ノードの増加に伴いキャプチャ作業を行うノードも増加するため作業量は増加する．

(2) ミラーリング

計算機ノード間の中継機器であるルータやスイッチの持つミラーリング機能を用いて，その機器を通過したパケットの情報を得られる．一般的に，ミラーリング機能は通信トラフィックが増大した際にパケットのロスが発生する．

(3) プロキシ

ウェブプロキシに代表されるような，計算機ノードとのコネクションを終端して中継するプロキシノードが利用可能である場合，そのプロキシノード上でのキャプチャによりパケット情報を得られる．

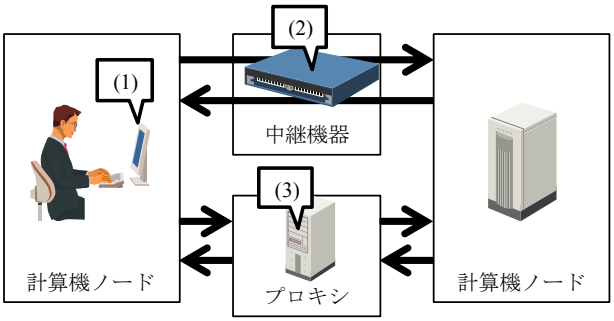

図 2　パケット情報収集方法

6. リプレイツール PARROT

本章では，テスト自動化のためのネットワークリプレイツール PARROT（Packet Analysis and Respondent Replay Operation Toolset）について述べる．

6.1 課題解決のためのデータ構造とアーキテクチャ

本節では 4 章で述べた解決方法の具体的なアーキテクチャについて述べる．

まず，PARROT ではメッセージ情報の単位で情報を保持することで解析およびリプレイの処理が簡潔となると考え，メッセージと対応するクラスである Message クラスを定義した．続いて以下に各課題の解決方法について述べる．

(1) メッセージ情報

メッセージ情報は，解析結果の確認および編集を容易とするために XML 形式で, Message インスタンス単位で出力する．図 3 にメッセージ XML の例を示す．プロトコルごとに情報を出力し，人的および機械的いずれにおいても情報の取得と編集を容易としている．

また，リプレイ結果のログもメッセージ情報のフォーマットを採用し，実装規模の低減を行う．

```
<message id="request">
  <protocol name="ethernet" order="0">
    <src>00:11:22:33:44:55</src>
    <dst>aa:bb:cc:dd:ee:ff</dst>
  </protocol>
  <protocol name="ipv4" order="1">
    <src>192.168.0.100</src>
    <dst>192.168.0.200</dst>
  </protocol>
  <protocol name="tcp" order="2">
    <src>12345</src>
    <dst>54321</dst>
  </protocol>
  <protocol name="ap" order="3">
    <data>"application data"</data>
  </protocol>
</message>
```

図 3　メッセージ XML の例

(2) スケジュール情報

スケジュール情報もメッセージ情報と同様に XML 形式とする．メッセージリプレイに必要な契機や動作の定義は，状態の考え方を導入することでステートマシンとしてモデル化できる．例として，リクエスト受信後 10 ミリ秒後にレスポンスを返すようなスケジュールは図 4 のステートマシンのように表現し情報は図 5 の XML 形式となる．

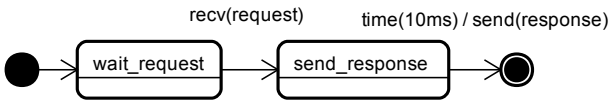

図 4　ステートマシンの例

```
<state id="wait_request">
  <transition id="send_response">
    <trigger type="recv" val="request"/>
  </transition>
</state>
<state id="send_response">
  <transition id="end">
    <trigger type="time" val="10ms"/>
    <action type="send" val="response"/>
  </transition>
</state>
<state id="end"/>
```

図 5　スケジュール XML の例

(3) **プロトコル拡張の容易化**

PARROTにおけるプロトコル処理の拡張性向上のためには，新たなプロトコルを実装する際の開発規模の抑制が必要である．そのために以下の 3 点を要点とする．

通信プロトコルクラス間での疎結合

各通信プロトコル間の密結合化は循環参照や内部構造の複雑化を招くため避けるべきである．一方でプロトコル処理のためにはパケットや他プロトコルの情報を得る手段が提供される必要がある．

プロトコル処理クラスと情報を持つクラスを分離

プロトコル処理は集中的に行わなければならない一方で，プロトコルの処理単位であるセッションごとに情報の保持が必要であるため，この 2 つは異なるクラスに実装されるべきである．

プロトコル処理共通化

PARROT では，プロトコルごとのパケットの情報の処理は，パケット情報からメッセージ情報を作成する際と，リプレイにおいてテスト対象ノードからパケットを受信した際に行う．この 2 通りのプロトコル処理内容は基本的に同一であるためルーチンを共通化することが望ましい．

疎結合を実現する設計の参考として GoF デザインパターン[8]の Mediator パターンを参考に設計した．結果を図 6 に示す．プロトコル処理を行う Protocol クラスと，その処理単位である ProtocolSession クラスを定義し，パケットの情報とプロトコルを横断する情報を PacketInformation クラスが保持することで要求項目を実現する．また解析結果はプロトコル単位の処理結果の集合であるため Message クラスで ProtocolSession インスタンスを集約する．

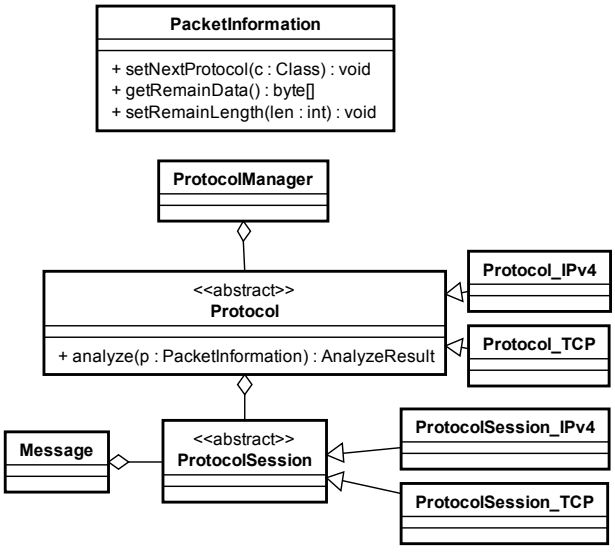

図 6　パケット情報とプロトコル

(4) **拡張機能**

PARROTユーザの拡張によるメッセージへのアクセスを実現するため，メッセージに関する処理の中心となるインタフェースクラスを用意する．アプリケーションはインタフェースクラスを実装して，にハンドラを一元管理するクラスに自身を登録する．

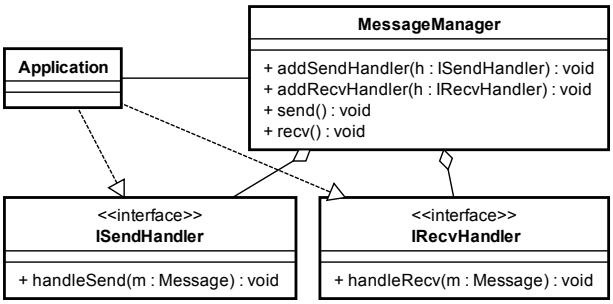

図 7　ハンドラの集約

6.2 PARROT 概要

本節では PARROT の全体概要について示す．PARROT の構成図を図 8 に示す．

PARROT の利用は 2 つのフェイズに分かれる．第 1 の解析フェイズは，収集したパケット情報を入力として解析を行い，6.1(1)で示したメッセージ情報と，6.1(2)で示したスケジュール情報を出力する．第 2 のリプレイフェイズは，リプレイ情報を入力としてリプレイ送受信処理を行う．解析フェイズおよびリプレイフェイズ双方においてプロトコルごとの解析処理が必要であるが，6.1(3)の方式により処理ルーチンは共通化している．

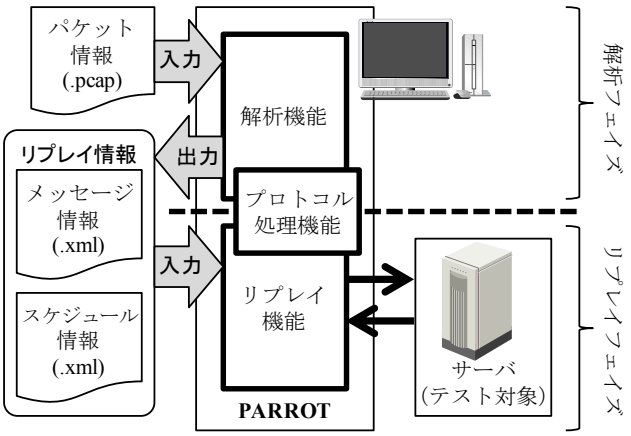

図 8　PARROT 概要

6.3 編集項目

PARROTにおけるリプレイにおける柔軟性向上のための情報編集について述べる．表 1 に編集項目を示す．編集方法は 3 通りあり，1 つ目は 6.1 (1)および(2)で示した xml ベースのリプレイ情報の編集である．2 つ目は 6.1 (4)で示した拡張であり，PARROT の実装プログラム言語である java を用いる．3 つ目は，PARROT が実行時に読み込むテキス

トベースの設定ファイルに記述する方法である．

編集項目はメッセージの内容のみならず，送信の時間間隔や順序，リプレイ速度を変更できるため，キャプチャ時には人手作業で行われていた作業の時間を飛ばしてリプレイを行える．

表 1　編集項目

編集項目	xml	java	txt
メッセージ内容	✔	✔	―
メッセージ送信時間/間隔	✔	―	―
メッセージ送信順序/繰り返し	✔	―	―
リプレイ速度（早送り，遅回し）	―	―	✔
リプレイ先 IP アドレスの変更	✔	―	✔

凡例　✔:編集可能　―:編集不可能

7. PARROT の適用

本章では PARROT を適用した自動テストについて，システムの概要および課題を示し，解決のための拡張について述べる．

7.1 適用対象システム

PARROT によるリプレイテストを適用したシステムについて概要を図 10 に示す．対象システムは，オペレータが操作するクライアントから送信したリクエストに対してサーバがレスポンスを返す処理方式であり，TCP/IP ベースのオリジナルプロトコルにより通信を行う．

これまで，対象システムにおいてサーバをテストする際には，クライアントを人手で操作して得られたレスポンスがシステムの仕様に合致しているかは画面の目視およびログにより確認していた．

対象システムにおいて問題となったのは，システムの改修に伴うリグレッションテストである．理想的には，全種類のリクエストに対して正しいレスポンスが返されるか確認するべきだが，種類は全体で 300 万件あり人手で作業を行った場合には 1 件あたり 10 秒としても 1 年以上を要するため現実的ではない．サーバ単体の試験であればシェル等を用いて実施できるが，本テストの要件であるリクエストに対するレスポンスの確認という点で不十分である．

そこで，リプレイテストがこれら問題を解決する手段として検討，採用された．

7.2 テスト自動化方針

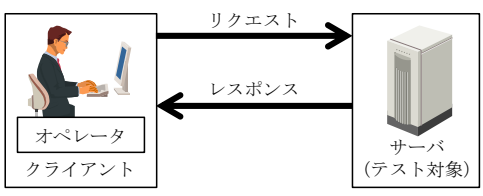

図 10　対象システム概要

PARROT を用いた自動リプレイテストにより対象システムでの 300 万件のテストを実現する．PARROT によりクライアントを模擬し，人手作業の操作に代わって自動でリクエストの送信およびレスポンスの受信を行う．

PARROT により受信したレスポンスの内容は得られるが，PARROT の仕様上，クライアント端末に表示されるであろう画面の作成は困難であるが，レスポンスに対する画面表示の整合性を確認するテストは別途行われているため，本適用においては，レスポンスの内容が仕様通りであれば画面の確認は必要ないとする．このレスポンスの内容の処理は別途検証用ツールを用いる方針とし，PARROT は検証ツールに必要なリプレイ結果情報の出力までを行う．

これら機能の実装における課題は，テスト自動化に伴う工数の抑制による適用効果の最大化である．そのために，1 件のリプレイ情報から 300 万件のリプレイを行うような拡張を行う方針とする．

7.3 拡張内容

PARROT 適用のための拡張内容について図 9 に示す．

(1) バリエーション情報

自動実行するリクエスト内容をバリエーション情報として定義する．ファイル形式は CSV として 1 行に 1 つのリクエストに関する情報を記述する．

(2) リプレイ結果

リプレイによる自動実行の結果を出力を定義する．ファイル形式は CSV として 1 行に 1 つのレスポンスに関する情報を出力する．

(3) スケジュール情報の編集

スケジュール情報のうち 2 点を編集する．1 点目は連続

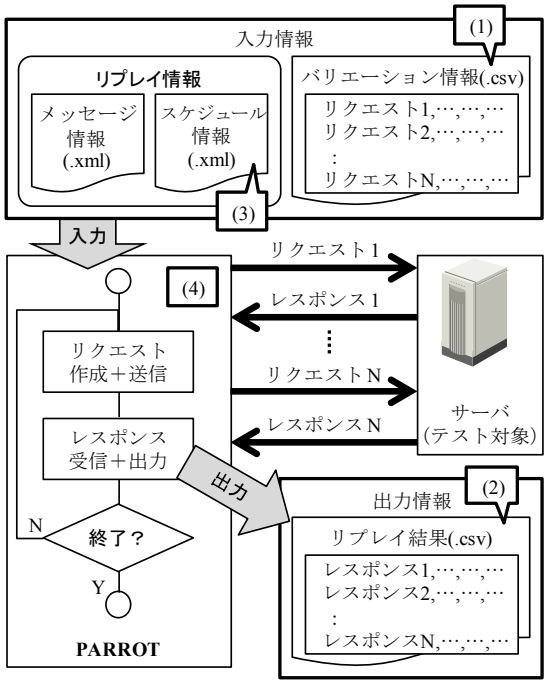

図 9　PARROT の拡張と適用

した実施のためにリクエスト送信とレスポンス受信の処理を繰り返すようにする．2 点目は，キャプチャ時にクライアント側で発生していた人手作業による待機時間を削除して即座にリクエストを送信するようにする．

(4) **拡張機能による連続実行ルーチンの導入**

リプレイ情報に基づくリプレイに加えて拡張機能により，バリエーション情報に基づいて内容を編集したリクエストを送信し，受信したレスポンスの内容をリプレイ結果に出力する．

7.4 結果

本節では PARROT の実装と適用の結果を示す．表 2 に示すように，実装規模は PARROT 全体の約 9k ステップに対して適用システム向けのプロトコル対応とアプリケーション部分で約 1.2k ステップとなった．プロトコル対応部分はアプリケーションに依存しておらず，今後同様のプロトコルに対応するアプリケーションを作成する際には再利用可能である．

表 2　Java 言語による実装規模

	ファイル数	実効ステップ
アプリケーション	1	218
独自プロトコル対応	9	988
全体	94	9089

適用に先立ち，結果検討のために PARROT の限界性能値を測定する．あらかじめ PARROT を 2 台起動してそれぞれサーバとクライアントをリプレイさせてその性能を測定した．

収集したリプレイ情報を用いてリプレイテストにより合計 300 万件のテストを自動実行した．適用結果と PARROT の限界性能値と比較すると，性能は限界値の約 15%に留まった．この性能の低下分はサーバ側の応答を待つ時間であり，PARROT が単一クライアントに対するサーバ性能の限界近くまでリプレイを行えたことを意味する．

7.5 評価

PARROT によるリプレイテストでは，1 件のテストに相当するキャプチャパケットから 300 万件のメッセージを作成し，サーバの性能を限界近くまで引き出す効率的な自動実行を実現した．また，クライアントを人手で操作する際と同じメッセージ内容でリプレイテストが行えるため，テストのためにサーバに手を加える必要はなく，準備作業による作業工数の増加は軽微であった．以上から PARROT を用いたリプレイテストにより有意義な規模のテスト工数の削減ができたと言える．

本適用ではサーバの性能を引き出して効率的に自動実行を実現した．これは，PARROT では実際のクライアントの処理を行うわけではなく，そのメッセージ送受信の再現処理を行うため，PARROT を実行する計算機の負荷が小さいためと考えられる．これは少ない機器で大きな負荷を発生させられることを示しており，今後は負荷テストへの応用も考えられる．

8. 今後の課題

本稿で示した適用では，あらかじめキャプチャした 1 つのメッセージ送受信フローのリプレイである．異なるフローへ対応する際にはメッセージ情報およびスケジュール情報を手作業で編集する必要があり作業コストの点で課題が残る．

また，本稿でのリプレイにより実現したデータバリエーションテストは現実的に実行可能な規模であったが，システムによっては全てのバリエーションのテストが現実的でない場合もある．今後はソフトウェアの仕様やアーキテクチャ技術と連携しテストの妥当性向上についても検討を行いたい．

9. おわりに

本稿では，リプレイテストのためのソフトウェアおよびその適用について述べた．要件だけでなくソフトウェアアーキテクチャ関しても検討した結果，開発コストの増大防止と再利用性の向上を実現した．実際のシステムに対する適用では計画通りのテストを自動実行し，良好な結果を得た．

今後はプロトコルの拡充などの機能追加に加えてテストデータの作成技術やテスト品質の向上技術についても検討を行う予定である．

参考文献

1) Cem Kaner, Jack Falk, Hung Quoc Nguyen（著）, テスト技術者交流会（訳）, 基本から学ぶソフトウェアテスト,
2) D. Williams, IT オペレーションにおけるスクリプト活用による自動化：混沌状態を回避するには, Gartner Research, December 3, 2010.
3) CA LISA, Service Virtualization, Online at http://www.ca.com/jp/devcenter/ca-lisa.aspx.
4) Oracle, Database Replay, Online at http://docs.oracle.com/cd/B28359_01/server.111/e12253/dbr_intro.htm.
5) Martin Fowler, "Mocks Aren't Stubs", Online at http://www.martinfowler.com/articles/mocksArentStubs.html
6) tcpdump, Online at http://www.tcpdump.org
7) Wireshark, Online at https://www.wireshark.org
8) Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides（著）, 本位田真一, 吉田和樹（監訳）, オブジェクト指向における再利用のためのデザインパターン改訂版, ソフトバンクパブリッシング, 2000.