ネットワーク実験支援ソフトウェアの汎用アーキテクチャの提案

宮地利幸[†1,†2] 三輪信介[†3]
知念賢一[†1,†2] 篠田陽一[†2,†3,†4]

実環境で動作するハードウェアおよびソフトウェアを用いて実験を行う環境では，手作業での実験用の環境構築および，実験制御は困難であるばかりでなく，実験の精度を低下させる．このような問題はノード数が多くなるほど顕著である．この問題を解決するため，自動的に実験を行うための前処理や，手順通に実験を実行する実験支援ソフトウェアが利用されている．現在，様々な目的を持った実験用設備が存在し，それぞれに専用の支援ソフトウェアが用意されている．それぞれの支援ソフトウェアは対象の実験設備に特化して設計されており，その性能を最大限引き出すことができる．しかしその一方，1 つの実験支援ソフトウェアでこれらの環境を協調動作させた場合，支援ソフトウェアは複数の実験用設備の平均的な機能しか利用できない．そこで，支援ソフトウェアの汎用アーキテクチャを定義し，それに従って，実験支援ソフトウェアを設計・実装することにより，他の実験設備で動作する実験支援ソフトウェアとの協調が可能となり，より柔軟に実験用の環境を構築できる．本論文では，実験支援ソフトウェアに必要な機能をまとめ，それを実現するための汎用アーキテクチャを提案する．我々の提案する汎用アーキテクチャにより，実験に必要な機能群を実現できる．

A Generic Architecture of Supporting Software for Network Experiments

Toshiyuki Miyachi,[†1,†2] Shinsuke Miwa,[†3] Ken-ichi Chinen[†1,†2] and Yoichi Shinoda[†2,†3,†4]

Building and administrating a testbed using an actual implementation by manual operation presents difficulties. One is the cost problem. A user must control many nodes to build an environment for experiments, therefore a user should log in and execute commands on all nodes. Thus much time and human resources are spent. Another problem is the accuracy of the experiment results. Manual operations are affected by inaccurate human behavior, which will have impacts on the results of the experiment. These problems are solved by using supporting software for experiment which drives experiments automatically. There are testbeds which have special purpose. To make an experiment on these testbeds, a special supporting software is needed because the supporting software should permit to use all the functions of the testbed. Simultaneously using multiple testbeds enables users to create a flexible environment for experiments. In this case, supporting software on each testbed should communicate to drive the experiment. Therefore we need a general architecture for these supporting software. With supporting software based on our architecture, we can use all the functions of multiple testbeds. In this paper, we describe the required functions for general supporting software, and propose an architecture to satisfy these functions.

1. はじめに

新たなネットワークソフトウェア実装をインターネットなどの実環境に導入するためには，導入以前に十分な検証を行う必要がある．このような実装の検証は，最終的な導入先である対象の環境自体を用いることが理想であり，このような検証の結果の信頼性は非常に高い．これまでインターネットは開発環境であった

†1 北陸先端科学技術大学院大学情報科学研究科/インターネット研究センター
School of Information Science/Internet Research Center, Japan Advanced Institute of Science and Technology
†2 情報通信研究機構北陸リサーチセンター
Hokuriku Research Center, National Institute of Information and Communications Technology
†3 情報通信研究機構情報通信セキュリティ研究センター
Information Security Research Center, National Institute of Information and Communications Technology
†4 北陸先端科学技術大学院大学情報科学センター/インターネット研究センター
Center for Information Science/Internet Research Center, Japan Advanced Institute of Science and Technology

ため，インターネット自体での検証実験が基本であった．しかし，現状のインターネットでは，様々な商用のサービスが行われている．このような，現状のインターネット上で実験を行えば，通信障害などを引き起こす可能性があるため，一利用者の判断で実験を行うことは危険であるといえる．これは，インターネット以外の実環境でも同様である．この問題を解決するため，実環境を模倣し，実験専用の環境を構築し，この環境を用いて実験が行われている．このような環境は，実環境で利用される PC やネットワーク機器とその上で動作する汎用的なソフトウェアを使いインターネットを模倣した環境と，ソフトウェアにより実環境および対象技術を抽象化し，その動作を検証するソフトウェアシミュレーションに大別される．

本論文では，実験用のハードウェア設備を実験設備，実験設備上での実験を行うために実験実行者を支援するソフトウェアを実験支援ソフトウェア，実験設備と実験支援ソフトウェアが用意された環境を実証環境と呼ぶ．また，ソフトウェアシミュレータや実証環境といった，実験を行うための基盤環境を実験実行環境とする．実験実行者は，実験実行環境上のネットワーク機器および PC などのユーザ端末などのノードを利用して実験用の環境を構築して実験を行う．ある実験のために実証環境上に構築された，実験トポロジおよびそれを構成するノード上で動作するアプリケーション，実験を管理するために必要なアプリケーションなどを含めた実験用の環境を，実験駆動単位と呼ぶ．また，実環境で利用されているものと同様のノードを実ノードと呼ぶ．

我々は，インターネットで利用される実装を検証することを目的とし，多数の実ノードによる実験専用の実験設備を用意し，これらを利用し容易に実験を議論を進めてきた．また，このような環境の一実装として StarBED および VM Nebula を提案・実装し運用している．StarBED は，多数の実ノードからなる実験設備であり，StarBED 上での実験を補助するための SpringOS も開発・運用している．

これまで我々は，StarBED および SpringOS の開発およびその運用を通して，様々な経験を積んできた．本論文では，これまでの経験から得られた知見および，複数の実験実行者へのインタビュー結果による支援ソフトウェアに必要な機能を述べ，これを満たすための支援ソフトウェアの汎用アーキテクチャを提案する．StarBED および SpringOS については，文献 1)，2) に，VM Nebula については文献 3) にまとめられている．

2. 実証環境への要求

本章では，実験実行者が，実験設備および支援ソフトウェアに対して求める要求について議論する．

ネットワーク技術の研究施設では，用意できるノードの台数の制限や，各ノードの制御のしやすさから，20 台から 30 台程度までの実験ノードを用いた実験用の環境は，現状でも頻繁に利用されている．しかし，これ以上の実験要素を持つ実験用の環境は，ノードの設定および制御が困難であるため，これ以上のノードを利用した実験例はあまり多くない．しかし，数千台のノードにより構成される実環境も珍しくなく，たとえばインターネットは数億台のノードから構成されるといわれる．ネットワーク技術の検証では，対象技術の導入先と実験用の環境の同一性が重要であるが，少なくとも規模に関しては，実験用の環境と実環境の規模は大きく異なる．

前述のような小規模な実験用の環境で検証された実装を，大規模な実環境へ導入した場合，その規模および複雑さの差異により，検証時点では観測できなかった問題が発生する可能性がある．このような問題を，導入前の検証で発見し，実環境に対する想定外の影響を低減することが我々の目的の 1 つである．したがって，本論文では現状で構築することが比較的困難である，数百台から数千台規模の実験用の環境を容易に構築できることを念頭に要求を整理する．

我々は，実験実行者にインタビューを行い，実証環境に必要とされる機能について検討を行った．本章では，実ノードを利用した一般的な実験手順についてまとめ，インタビューの内容およびその意図を説明し，インタビュー結果から洗い出された要望を示す．

2.1 一般的な実験手順

まず，実ノードを利用した実験の一般的な手順を確認する．

1) **実験トポロジやシナリオの検討** 実験の目的により適切なネットワークトポロジや各ノードに導入する OS および必要なアプリケーションを検討し，さらにその環境上で実行されるべきシナリオを検討する．
2) **必要なノードなどの用意** 検討の結果から必要なノードを用意する．
3) **各機器の接続** 用意したノードを接続する．
4) **ノードへのソフトウェアの導入** ノードへ必要な OS やアプリケーションを導入し，必要な設定を行う．
5) **構築した環境上でのシナリオ実行** 構築された環

境上で実験シナリオを遂行し，実験データを収集する．

6) ログの解析 得られた実験データを解析する．

ここでは，それぞれの手順が一度のみ実行される場合を示したが，一度構築した実験駆動単位を利用し，アプリケーションのパラメータなどの変更のみで，複数回シナリオを実行することはネットワーク実験においては一般的である．

以上であげた手順のうち，実験環境の構築および実験シナリオの実行に関わる手順 2) から 5) を本論文での対象とする．

2.2 実験実行者へのインタビュー

我々が，これまでの StarBED および SpringOS，VM Nebula の開発・運用を行ってきた経験と，複数の実験実行者に対して行ったインタビューの結果から，実証環境に対する要望を整理する．インタビューは，実証環境やソフトウェアシミュレータでの 15 程度の実験について，組織内外の研究者に対して行った．実験の内容は，プロトコルの論理的性能把握や，ハードウェアおよびソフトウェア実装の単体検証，性能検証，導入環境を想定した比較的大規模な環境での影響試験などが目的であった．インタビューの内容と意図を以下にまとめる．

検証技術概要 検証する技術の性質などを確認する．

実験フェーズ ソフトウェア開発では，アイデアの提案から，実環境に技術を導入するまでに様々なフェーズがある．対象実験がソフトウェア開発のどのフェーズに位置するかにより，実験自体の目的が異なるため，要求事項は異なる．

実験スケジュール 対象実験以前に行った実験と，その後に行った実験を確認することで，実験の目的およびその結果をより詳細に検討する．

実験トポロジ どのようなトポロジを用いて実験を行ったかを知ることにより，実験トポロジを構築するための要求を検討する．

検証項目 どのような項目についての情報を集めたかを確認することにより，実験手順を実行するうえでの要求を導出する．

トラフィックパターン 実験では何らかのトラフィックを発生させる．どのようなパターンのトラフィックを利用したかを確認することにより，ノード制御への要求を明らかにする．

2.3 実証環境への要望整理

インタビューの結果，基本的には実験を自動的かつ柔軟に行えることが求められるほか，それ以外にも様々な要求があることが分かった．これらの結果と，我々の運用・開発経験から導かれる要求を以下にまとめる．

設定記述に従った自動的な実験実行 実験実行者の理想は，実験対象の挙動や性能を知ることであり，実験の実行自体に労力をかけることは本質ではない．また，実験実行者が実験を行うために必要な手順をすべて手作業で行う場合，実験実行に必要な時間および手間は大きい．これは実験用のトポロジをつくることが困難であるだけでなく，実験シナリオを実行のためには，つねに各ノードの状態を確認し，コマンドを実行しなければならないためである．また，人の手作業によるシナリオ実行は，精度を低下させるおそれもあるため，このような機構が必要となる．

前もって実験実行者が記述した実験内容に従い，支援ソフトウェアが実験を自動的に進められれば，実験実行者の，実験の準備および，実験実行にかかる手間を軽減できるだけでなく，機械的に実験の準備および実行を行うことで，実験結果の精度低下も期待できる．

実験トポロジの柔軟な設定 実験実行者が，実験の対象となるトポロジを構築するため，実証環境に存在するノードの配置などを検討することは対象のトポロジが大規模になるほど手間がかかる作業である．実験記述に従って自動的に必要な機能を持つノードを割り当て，目的のトポロジを構築できることが要求されている．トポロジの変更や手作業によるスイッチの設定なども自動的に行うことで，実験実行の手間の軽減および実験精度の向上が期待できる．また，ベンチマークに関する実験などでは，測定対象のリンクの帯域やジッタなどを制限したいという要求が強い．このような要求を満たすため，設定に記述されたリンク特性の自動設定も求められている．

ノードへの自動的なアプリケーション導入および設定 実験実行者は実験に利用するノードへ，必要な OS やアプリケーションなどのインストールや，ネットワークやアプリケーションについての設定を行う必要がある．このような作業は，多くのノードに対して同様の設定を行うことが必要であり，効率的とはいえない．この手間を低減させるため，実験設定に記述された内容に従い，ノードへ必要な OS やアプリケーションの導入および設定の自動化が求められる．

シナリオの自動実行 実験実行者の意図する順序・タイミングで，ノード上でコマンドを実行すること

により実験シナリオを実行できることが求められる．前述のとおり，これにより実験実行の手間の低減だけでなく，精度の再現性の向上も期待できる．

ノードの時刻の同期機構 実験のログを各ノードに残す場合は，そのタイムスタンプで実験の経過を確認することとなる．このような場合にノード時刻が正しく同期できていないと，実験実行後の実験結果の検証に支障をきたすおそれがある．したがって何らかの方法でのノード時刻の同期が求められる．ただし，実際の時刻に合わせる必要はなく，実験駆動単位に存在するノードで時間が正確に同期できていればよい．

実験状態の把握機構 実験実行中には各ノードやトラフィック，トポロジの状況を把握することで，実験の進行状況や，トラブルの発生状況を確認できる．このような機能がない場合は，想定外の状況が発生しても，実験実行後，結果を確認しなければその状況が把握できず，実験自体が無駄になる可能性もあるため，容易に実験駆動単位の情報を取得できることが求められる．

ログの自動収集 実験実行者は，実験実行後に実験用ノードのログを収集し，その結果から対象技術の分析を行う．多数のノードによる環境を利用した場合は，収集すべきログは多数のノードに分散していることも多く，ログを収集する手間が大きい．この手間を低減させるための自動的なログ収集が要求される．

実証環境の初期化 実験駆動単位を構築するためには，実証環境のノードに対して様々な設定を行うことになる．必要な OS やファームウェアおよびアプリケーションのインストールや，IP アドレスなどのネットワーク関連の設定がその例である．このような設定が，次の実験を実行する際に残っていると障害の原因となりかねない．したがって，実験が終わったタイミングで実証環境を自動的な初期化が求められている．

他の実験実行者との資源分離 複数の実験実行者で実証環境を共有し，それぞれの実験駆動単位を構築することがある．このような場合には，他の実験に影響を及ぼさないよう，資源を分離したいという要求がある．

実験状態の保存 実験途中に施設の予約期間が終わってしまった場合などに，ある実験の状態を保存したいという要求がある．

複数の実験環境の接続 実証環境には様々な特性を持ったものがある．たとえば，StarBED は特定のネットワーク実験に特化したものではなく，汎用的な実験設備であり，様々な用途に利用できる．一方 SIOS[4] や VM Nebula といった実証環境は，StarBED と異なりセキュリティ関連の実験に特化されて設計されている．これらの実証環境群を，実験トポロジの適切な部分に配置し利用することで，実験トポロジの規模の向上だけではなく，実験実行者の意図する実験を，より柔軟に実行できる[5),6)]．このような実験を容易に実現するため，実証環境を相互に接続し，透過的に操作したいという要望がある．

現実的な実験トポロジ 実験駆動単位を構築する際に，現実的な実験トポロジを容易に利用できることが要望としてあがっている．しかしこれについては，トポロジのモデリングについての議論であり，本論文での汎用的な実験支援ソフトウェアの範疇からはずれる議論であるため，本論文では対象としない．ただし，現実的な実験トポロジを定義した場合に，それを容易に実証環境上に実現できる柔軟さを持った支援ソフトウェアを設計することが必要である．

現実的なバックグラウンドトラフィック 現実的なバックグラウンドトラフィックについても，容易に生成・導入できる仕組みが求められている．実際のトラフィックデータを利用し実験導入することは可能であるが，実験により利用できるトラフィックの種類は変化する．これについても，支援ソフトウェアの議論とは別に，現実的なバックグラウンドトラフィックの生成や導入に関する議論が必要である．ただし，定義されたトラフィックを容易に導入できる仕組みは必要である．

以上が，実証環境に求められる機能である．これらの要求を満たすことにより，実験実行者が実験を行うために必要な工数を削減するとともに，より精度の高い実験が可能となる．

3. 実証環境に求められる機能の実現

本章では，まず我々が前提とする実証環境について述べ，2 章で述べた実験実行者の要求を満たすため要素機能についてまとめる．

3.1 前提事項

まず我々が前提とする実証環境は，複数の実験実行者で同時に利用されることを前提とする．したがって，1 つの実証環境に複数の実験駆動単位が生成される．

実験実行者は，実験実行に必要な資源を実証環境に

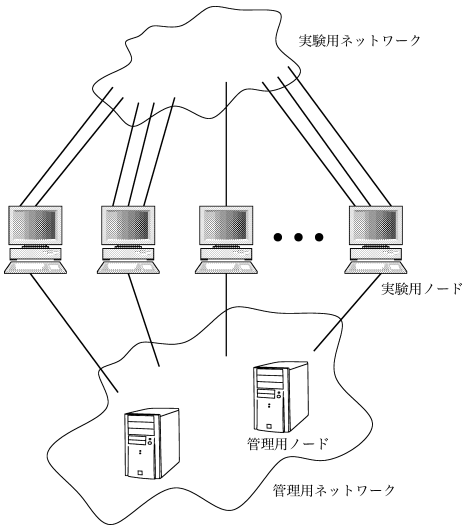

図 1 我々が想定する実証環境のトポロジ
Fig. 1 Assumed topology of the testbed.

要求し，割当てを受けその資源をもちいて実験駆動単位を構築する．

また，実験駆動単位を構成するノードのネットワーク設定は，実験実行者の意図により決定される．この場合，実験駆動単位を設定中は，ネットワーク設定が行われていないため，ノード設定の実行は不可能である．したがって，我々が想定する実証環境では，管理用のネットワークが用意され，すべてのノードに前もってネットワーク設定が行われていることとする．**図 1** に我々が前提とするトポロジの概念を示す．

3.2 資源管理および資源割当て

実験実行のためには，実証環境に存在する資源を自動的にあてはめる必要がある．このためには，実証環境に存在する資源の情報を管理し，実験実行者に割り当てる必要がある．本論文ではこの機能を持つモジュールをリソースマネージャと呼ぶ．

リソースマネージャが管理するべき資源は，実証環境の実装にも依存するが，ノード，帯域などのリンク特性などがあり，実験ネットワークを VLAN[7] で構築する場合は，VLAN 番号も資源情報として管理される．資源の種類は実証環境によって異なる．また実験駆動単位の制御機構が動作するノードも資源として管理される．

リソースマネージャには，実証環境の資源を管理するだけでなく，資源を実験駆動単位または実験実行者に割り当てる機能も必要である．実験実行者は設定記述として，それぞれのノードやリンクに必要とする機能を記述する．この実験設定に従い，実験駆動単位制御モジュールが，実証環境のリソースマネージャに割当て要求を行う．リソースマネージャはこの要求を満たすノードを何らかのアルゴリズムで選択・割り当てる．また，基本的には割り当てたノードを別の実験実行者に割り当てないよう制御を行う．

実証環境の資源によっては，一実験実行者に占有される場合ばかりではなく，複数の実験実行者に共有される可能性がある．これにより，実証環境上の資源をより効率的に利用できる．このためには，実験実行者が必要としている資源に応じて共有できるかどうか，共有できるのであればその程度などが考慮されなければならない．ノードが共有される場合は，CPU やメモリの量などから共有できるかなどからその可否が判断される．またリンクについては必要な帯域や，許容できる遅延などの情報をもとに共有できるかで判断される．実証環境によっては，リソースマネージャが，このような資源共有のためのアルゴリズムを持っている必要がある．ただし，実ノードを用いている以上，指定した値でノード負荷やネットワーク特性を固定することは不可能であるため，実験実行者による機能要求は，ある程度の範囲での指定を行うこととする．

実証環境の利用方式として，利用する時点で資源が空いているかどうかを確認し，空いていれば実験実行するといったバッチ型の実行方式もあるが，ある期間で必要数のノードを確保しておき，実験を行う方式も考えられる．柔軟な実験制御のためには両方式に対応することが望ましい．資源をある期間，実験実行者に割り当てる場合は，予約状況も資源の情報として保持されなければならない．

3.3 資源利用の排他処理

ノードやリンクの割当ての排他制御についてはすでに述べたとおり，リソースマネージャが管理するが，その他の資源や機能についての排他制御が必要となる．共有される可能性のある資源の割当てやそのような資源に対する操作を行う機構では，排他制御を行う必要がある．

3.4 設定読み込みおよび実験駆動単位の生成

自動的に実験を実行するために，実験駆動単位を生成する必要がある．実験駆動単位は制御モジュールと実験用ノードで構成され，実験用ノードは制御モジュールにより施設から割り当てられ，管理される．したがって，実験駆動単位を構成するためには，まず，制御モジュール群を起動しなければならない．

このためには，制御モジュール群が動作するノードの割当てが必要である．割当て要求は入力された実験設定から，実験の規模などを考慮して行われ，リソースマネージャはそれを満たす機能を持つノードを割り

当てる．このとき，ノードに要求される性能は，実験の規模などによって異なるため，おおまかに実験の内容の読み込みを行う．

このようにして起動された制御モジュール群は実験設定を詳細に読み込み，実験資源を確保し，実験駆動単位を生成する．

3.5 ノード制御

前章であげられた多くの機能はノードを柔軟かつ自動的に制御することにより実現できる．本節ではこれらの機構についてまとめる．

3.5.1 ノードの死活管理

実験を行うために各ノードの電源投入が必要であり，実験終了時には電源を落とす機能が必要である．容易な操作で多くのノードの電源を操作できることが望ましい．また，実験準備の一部もしくは実験中にノードを再起動させる必要も生じることがあるため，再起動も行えることが必要である．

3.5.2 ノードへのソフトウェア導入および設定

ノードのディスクに対して，あらかじめ用意されたディスクイメージの書き込みを行ったり，既存の OS に対してのアプリケーション導入および，それらの設定を行えたりする必要がある．また，ハードディスクを利用してだけでなく，ディスクレスシステムとしてノードを動作させることにより，より時間的コストを軽減し，実験用ノードとして振る舞わせることができる．また，導入するべきディスクイメージの作成支援も必要である．

また，同じディスクイメージを利用した場合でも，アプリケーションのやネットワーク設定などのみ変えたい場合も多い．これについては，コマンドの自動実行機構により，必要なコマンドを実行することで各種設定を行うことができる．

3.5.3 トポロジおよびリンク特性の設定

指定されたトポロジ設定のため，自動的に実験トポロジを設定する仕組みと，各リンクの特性を変更できる仕組みが必要である．リンクの特性としては帯域やジッタなどがあげられる．

3.5.4 ノードでのコマンド実行

基本的に実験シナリオは各ノードでの実行されるコマンドの集合といえる．各ノード上でコマンドを自動的に実行するための機構を用意し，実験設定で記述されたコマンドを各ノードですることで実験シナリオを実行できる．このとき，複数のノードで同期してコマンドを実行する必要が生じる場合があるため，あるノードでのイベントをトリガにして，別のノードでコマンドを実行する仕組みが必要となる．

3.5.5 ノード時刻の同期機構

ノードの時刻を同期させるためには NTP [8] などを利用する方法が一般的であり，基本的に各ノード上でのコマンド実行により実現できる．したがって，非常に詳細な同期が必要な場合をのぞき，時刻同期に関して特別な機能を用意する必要はない．

3.5.6 実証環境の初期化

実証環境の初期化は，初期化の対象となるノードを利用した実験と見なすことができる．それぞれのノードの初期状態のディスクイメージや，設定ファイルをソフトウェアの導入機構を用いて書き込み，ノードを初期化する．また場合によってはコマンド実行機能での初期化も可能である．リソースマネージャと協調して動作することで，実験期間が終了した際に自動的に初期化を行うことも可能である．

3.5.7 実験実行者による手動実験制御

何らかのトラブルが発生した際に，実験実行者が，該当ノードに接続し，その操作を行いたいという要求は多い．実験実行者が実証環境の物理トポロジを意識せず，容易にノードのコンソールなどにアクセスできる機構が必要である．

3.6 実証環境および実験駆動単位の状態監視

実証環境としては，各ノードの電源の状況や管理ネットワークでの到達性などを検知できればその健全性を確認できる．また，実験駆動単位ではより詳細情報を管理したいという要求がある．各実験ノードの OS やシナリオ実行の履歴，ノードの各インタフェースの情報からトポロジ情報も取得することで，実験が正しく動作しているのかを確認できる．したがって，実証環境としてその健全性を確認する機構と，実験駆動単位で詳細に実験ノードやリンクの状況を確認する機構も必要である．様々な方法でノードの状況は確認できるが，これを実験実行者に分かりやすく提供する機能が必要となる．

3.7 ログ収集

ログは一般的には実験ノードにファイルに保存されるか，外部のノードに転送されまとめて保存される．どちらの場合でも，ノード上でコマンドの実行により，実験終了後に実験実行者が指定したノードへこれらのファイルを転送できればよい．したがってノード上でのコマンド実行機能があれば対応できる．

3.8 実験状態の保存

ある時点でノードのディスクイメージを保存することで，ディスクの状態を保存することができる．ただし，メモリや通信のためのコネクションの状態，さらには，リンク上のトラフィック状態を保存することは

困難である．ただし，実証環境の実装によりある程度は提供できる場合もあるため，実現方法は実証環境の実装に依存する．

3.9 複数の実証環境の協調

実験実行者とっては，それぞれの実証環境の特性や，それぞれの操作方法の詳細を学ぶことなく統一のインタフェース実験が行えることが望ましい．したがって，各実証環境で動作する各種機構が連携し，実験実行者には複数の実証環境を利用していることを意識させないインタフェースが必要である．

複数の実証環境にまたがる実験駆動単位を構築するためには，それぞれの実証環境に存在する制御機構の協調が必要であり，このために各実証環境に存在するモジュール間の一般的な通信方式が必要である．基本的には，以上までで述べた機能が実現できれば，実験駆動単位を構築できる．よほど特殊な機能を提供する実証環境以外は，これらの機能の実証環境内部での実現方法を隠蔽し，外部からは一般的なインタフェースでアクセスでき，複数の実証環境をまたがる実験駆動単位が構築できる．

3.10 例 外 処 理

以上であげた機能以外にも何らかの障害が起きた際に対応できる仕組みが必要である．実験対象の動作を事前に予測することは困難であり，実験中に，何らかの例外が発生することは，想像に難しくない．したがって，それぞれの機能に例外処理機構が必要である．

実証環境および実験駆動単位の状態監視機構と連携することで，ノードの状況に応じて何らかのイベントを駆動することが可能である．実験実行者は，状態監視機構で確認された状況により，実験実行者が処理内容を設定できる必要がある．このとき考えられる処理内容は，何らかのコマンドを実行する，実験を終了する，障害などを無視して実験を継続する，実験を一時停止し実験実行者からの指示を待つといった処理が考えられる．

4. 提案アーキテクチャ

2 章で洗い出した要求を実現するための要素技術を 3 章で述べた．本章ではこれらの機能を実現する汎用アーキテクチャを提案する．

本提案アーキテクチャでは，実験実行者が物理環境を意識することなく利用できることと，単一実証環境だけでなく，複数の実証環境を接続し，実験駆動単位を構築できることを念頭においた．このため，実験に必要な機能を提供するモジュールを，ある程度細かく分離し，各要素間の通信内容を定義する．また，実証環境間を結ぶ資源なども管理対象として，自動的に設定を行う．

以前の SpringOS では，実験駆動単位設定フェーズとシナリオ実行部分を分離し，それぞれで異なった機能を提供していた．この方式では，実験中のノード故障にともなう新たなノード割当てなどを行えない．本提案アーキテクチャでは，資源の獲得やノードの設定などは，実験中にも行えることとする．

機能別に大別して我々の提案するアーキテクチャを示す．

4.1 資 源 管 理

我々は，資源情報を 2 種類に分類した．1 つはノードなどの資源の静的な情報である．ノードの情報であれば，ネットワークインタフェースの種類や数，CPU，メモリ，ハードディスクなどハードウェア情報が含まれ，さらに，物理的なノードの ID や初期状態でインストールされている OS の情報，管理用のネットワークインタフェースのネットワーク設定も保持されるべきである．またリンクの情報であれば，メディアや帯域などが保持される．これらの情報は故障発生時などの例外的な場合にのみ書き換えられる．一方，資源の割当てなど動的な情報がある．ある資源がその時点で，どの実験もしくは実験実行者に割り当てられているのかといった情報は非常に頻繁に書き換えられる．また，ある実験実行者に割り当てることができる資源を管理する必要もある．これはいうなれば予約の管理である．

この 2 種類の資源情報を管理するモジュールを分離し，動的資源を管理するモジュールを DynamicResourceManager（DRM），静的資源を管理するモジュールを StaticResourceManager（SRM）とする．SRM はノードのハードウェア情報など変更頻度が低い情報を管理し，DRM は資源の予約状況や割当て情報および資源の状態を保持する．また，資源の状態を保持するのみではなく，実験実行者への資源の割当てを行う．また，実験実行者により持ち込まれた機材などを管理するため，リソースマネージャに，実験実行者により機材の情報を登録できるインタフェースも必要である．

リソースマネージャは実証環境全体の資源を管理し，実験実行者への資源割当てを調停する必要があるため，実証環境単位ごとに用意する必要がある．

4.2 設定読み込みおよび実験駆動単位の生成

実験駆動単位の制御モジュールを起動するノードを割り当てるため，おおまかな実験設定を読み込むためのモジュールを実証環境ごとに用意する．このモジュールを ExperimentScheduler（ES）と呼ぶ．ES

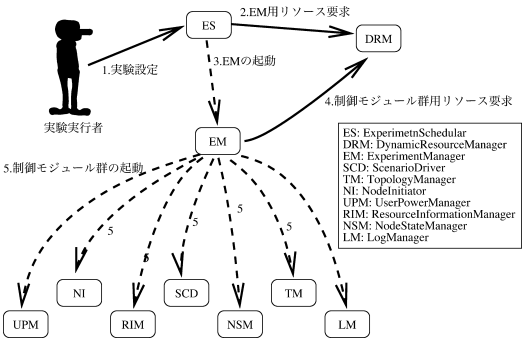

図 2 実験駆動単位の構築手順
Fig. 2 Steps for building the experimental unit.

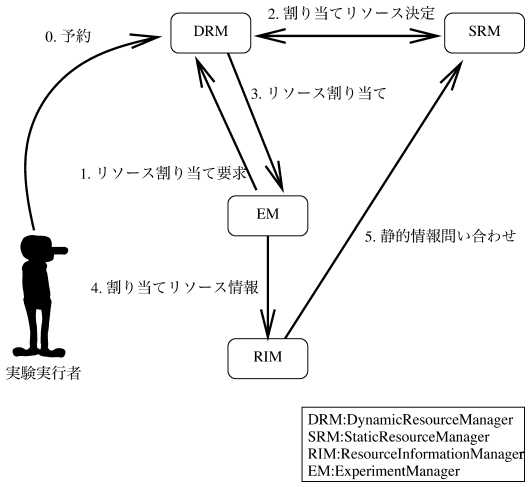

図 3 資源割当て手順
Fig. 3 Steps for allocating resources.

は実証環境のスケジューリングを行う機能を持ち，実験設定に基づき，指定されたタイミングで実験駆動単位を生成する．実験駆動単位には以後で説明する様々な制御モジュールが起動されるが，ES はこれらの制御モジュールを管理する ExperimentManager（EM）を起動する．具体的には ES は，EM が動作するノードの割当てを DRM に要求し，DRM は管理する資源を検索し，要求を満たす EM のためのノードが割当て可能であれば，ES にそれを割り当てる．ES はこのノード上で EM を起動する．起動された EM は ES からすべての設定ファイルを受け取り，より詳細に実験設定を解析，他の制御モジュールを起動するために，必要な性能を持つノードをそれぞれ要求する．EM 以外の制御モジュール群が起動すると，ES からノード設定や，シナリオ実行についての実験設定など，それぞれが必要とする部分の設定を受け取り，実験を実行する．

EM によって起動されるべきモジュールを，今後説明するものも含めて以下にあげる．また，**図 2** に動作手順を示す．

- ScenarioDriver：シナリオ駆動
- ResourceInformationManager：実験駆動単位に割り当てられたノード情報管理
- TopologyManager：実験駆動単位のトポロジ制御
- NodeStatusManager：ノード状態監視
- LogManager：ログ管理
- NodeInitiator：ノードへのソフトウェア導入および初期設定
- UserPowerManager：ノードの電源管理

4.3 ノード機能の指定および割当て

すでに制御用の資源割当てについて簡単に述べたが，実験用ノードも基本的に同様の手法で割り当てられる．制御用資源は実験規模などから必要性能を決定したが，実験用ノードの機能は実験記述に具体的に記述される．また物理ノード名などによる静的な指定も受け付けられる必要がある．

EM が DRM へ資源要求を行い，DRM がその条件を満たす資源の検索を行い，割当てを行う．割り当てられたという情報は DRM に保持されるが，より詳細な情報を保持するための制御モジュール ResourceInformationManager（RIM）に登録される．SRM，DRM は，実証環境の管理のためのモジュールであり，実験記述中で指定されたノードと物理ノードの対応や，動作すべき OS など，実験の詳細に関わる情報を持たないため，RIM がこのような情報を保持する．

図 3 に資源管理モジュール群の関連図を示す．図中 0 の予約が必要であるかどうかは実証環境の運用ポリシにより異なる．

4.4 ノードの死活管理

ノードの電源投入や電源断，再起動には，Magic Packet Technology [9] や IPMI [10] や iLO [11] を利用したハードウェアレベルでの実現方法と，SNMP [12)~14)] やコマンド実行による，ソフトウェアレベルでの実現方法がある．ハードウェアレベルでの制御を行うためには，実証環境のノードには，専用のハードウェアが実装されていなければならない．また，ソフトウェアで実現する場合は，クライアントプログラムが動作していればよいが，ノードの起動処理や，ハングアップ状態からの復帰処理を行うことはできない．

ノードの死活管理は，ノード上で動作する PowerAgent（PA）と，実験駆動単位の制御モジュールである UserPowerManager（UPM）で行われる．具体的には，PA は SNMP や IPMI などのクライアントである．UPM に対してあるノードの死活管理要求が行

われると，UPM が PA と通信し，要求された方式での死活管理を行う．死活管理を行う際に，PM は RIM に各ノードの状態を問い合わせ，他の実験駆動単位と共有されているノードの場合は，実証環境で動作する PowerManager（PM）に通信をし，調停のうえで処理を行う．

4.5 ノードへのソフトウェア導入および設定

ノードの起動方法は，ディスクに導入されている OS を利用する場合と，ディスクレスシステムとして起動する方法，そして，外部ノードから取得したディスクイメージをノードに導入し，導入した OS で起動する方法が考えられる．ノードのディスクに新たに OS を導入する場合は，ディスクレスシステムとして起動後，外部ノードからディスクイメージを取得し，ローカルディスクに書き込み，ノードをディスクから起動するよう変更して再起動すればよい．ディスクイメージの書き込みなどの処理は，ディスクレスシステムで動作するコマンド実行機構を利用すればよい．したがって，ディスクに導入されている OS からの起動と，ディスクレスシステムとしての起動を切り替える機構および，コマンド実行機構により，前述の 3 手法に対応できる．また，アプリケーションの設定や，ノードのネットワーク設定など基本的な設定は，後述のコマンド実行機能を用いて行うこととする．

起動方法切替えの具体的な実現例は，すべての PC ノードが起動時に必ず PXEboot でブートローダを取得し，それに従い起動する方法がある．この方法を用いると，各ノード用のブートローダを動的に変更することにより，PC ノードの起動方法を切り替えられる[15]．

それぞれのノード起動方法や，導入するディスクイメージなどは RIM に問い合わせる．ノードの起動方法は UserBootChanger（UBC）が変更する．たとえば，PXE でノードが起動する際に DHCP で取得するブートローダに関する情報を，LDAP やシンボリックリンクを利用することで変更する方法が考えられる．なお，他の実験とノードを共有している場合は，実証環境で動作する BootChanger（BC）が調停を行う．

ノードにソフトウェアを導入する際には，NodeInitiator（NI）が主体となって処理を進める．すべてのノード上では NI の指示によりコマンドを実行する，NodeAgent（NA）が動作しており，ノードに対する操作を行う．NI は NA および UBC，PM と協調しノードへソフトウェアを導入する．BC および NI は場合によってはファイルストレージを持っており，ブートイメージや必要なソフトウェアなどを保持する．

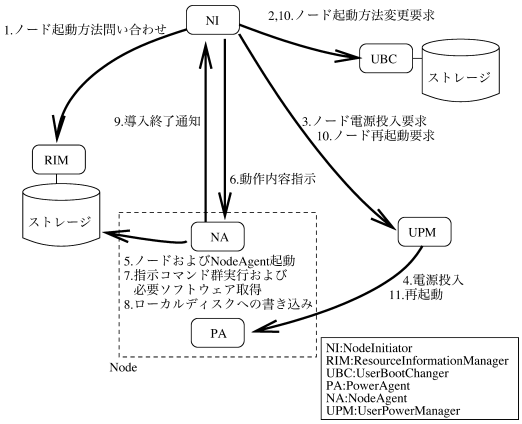

図 4 ノードのディスクへのソフトウェア導入
Fig. 4 Steps for introducing software to nodes.

ノードへのローカルディスクへの OS 導入手順を**図 4** 以下に示す．

ディスクイメージの生成方法としては，ハードディスクもしくは 1 パーティションの情報を，すべてファイルとして保存してしまう方法がある．また，ディスクレスシステムとして起動する場合は，それぞれの OS で専用の手法が用意されている場合が多い．PC ノードの場合はこういった手法が利用できるが，スイッチノードの場合は実装により様々である．

4.6 ノードでのコマンド実行

シナリオの実行は各ノード上でコマンドを実行することで進められる．各ノードでのシナリオの実行状態や，ノード間同期を実現するための ScenarioDriver（SCD）が EM から実験設定を受け取り，すべてのシナリオを管理する．各ノード上では，コマンドを実際に発行する ScenarioAgent（SCA）が動作する．SpringOS では，ノード起動後 SCD から SCA に対してそのノードのシナリオを送信し，SCA がそれに従って独立的にシナリオを進行させるモデルを採用している[16]．

各ノードの同期のため，SCA どうしが直接通信を行うと，各ノード上で管理されるべきセッション数が増加し，ノードの負荷が増大する．したがって，実験に利用するノードへの要求性能が大きくなる．これを解決するため，SCA は SCD を介してメッセージを交換する．これにより，各ノード上の SCA は，同期のための通信を SCD とだけ行えばいいため，管理セッション数は 1 つでよく，SCD が動作するノードのみある程度の性能を有していればよい．ただし，同期は様々な方式により実現可能であり，ここであげた方式以外でも，実証環境内で矛盾なくシナリオを実行できれば問題ない．

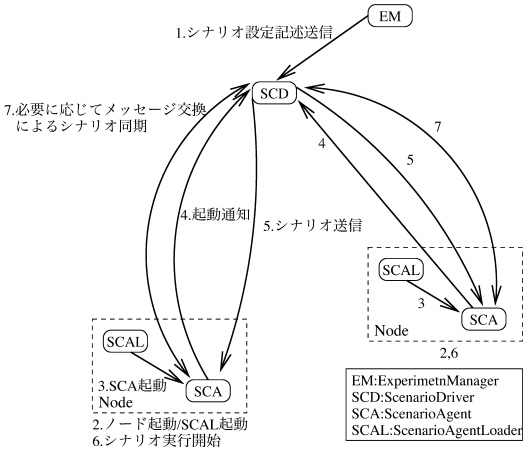

図 5 シナリオの実行手順
Fig. 5 Steps for executing scenario.

SCA の更新やバグに備え，SCA を起動するための ScenarioAgentLoader（SCAL）を用意する．SCAL は OS で提供される起動スクリプトや，非常にシンプルな機能を持った SCA が利用できる．

シナリオ実行の手順を**図 5** に示す．

4.7 トポロジおよびリンク特性の設定

トポロジを自動的に設定するため，実験設備には，ネットワークを利用して，外部から設定を変更することでトポロジを変更できるスイッチを導入する．VLAN や ATM の VP/VC による仮想的なネットワーク構成の変更や，物理的に配線を変更できるネットワーク機器が利用できる．また，VLAN を扱える PC ノードはこのようなスイッチノードと同様に利用することができる．

リンク特性の変更については，Dummynet [17] や NISTnet [18] などを利用すれば，特殊なノードを導入しなくても，PC ノード上でアプリケーションを実行するだけでよい．

実際の設定はスイッチノードでのシナリオ実行が行えればよい．リソースマネージャからトポロジ設定に必要な資源の割当てを受け，この資源を用いて各スイッチノードでコマンド実行による設定を行う．ただし，複数の実験駆動単位で共用されるスイッチノードでの VLAN などを用いたトポロジ変更や帯域制御に関する設定は，他の実験駆動単位との調停が必要であるため，実証環境に用意された SharedLinkManager（SLM）を通して行う．

4.8 実証環境および実験駆動単位の状態監視

ノード状態の監視方法として，SNMP や ICMP など一般的なプロトコルを利用する方法と，独自の方式を持ったクライアントを用いた方法がある．どちらの方法でも各ノード上で，専用のクライアントプログラムが動作している必要がある．このような情報の管理は，特に実験の詳細な情報を必要とせず，さらに基本的な情報から状態を検知しているため実証環境自体の機能として提供できる．ノードが異常な状態に陥っていた場合は，リソースマネージャに通知され，リソースマネージャから実験駆動単位の制御モジュールに通知される．

実験駆動単位では実験ノードやシナリオのより詳細な情報を保持しており，これらの情報から，状態管理もより詳細に行いたいという欲求がある．対象情報として，OS の種類やバージョン，実験シナリオの実行状態などがあげられる．これにより，実験の進行状態や，障害を検知することも可能である．これらは，実験駆動制御モジュールである StatusManager（SM）と，実験用ノード上で動作する StatusAgent（SA）により実現される．実証環境で動作する HealthChecker（HC）は基本的に電源の入切などおおまかな状態を監視する．しかし実験駆動単位では，SNMP などを利用したノードの CPU 負荷やネットワークインタフェースを流れるトラフィックなども管理の対象となる．また，SA と協調することにより，実験の進行状態も管理する．取得したノードの情報は RIM に反映される．

また，ノードだけではなく，設定したリンク特性の状態を知るために，トラフィックを発生させ帯域などを検証したり，設定通にトポロジが構築されているのかなどを検証したりする必要もある．これらは，ping などを利用してのノード間の疎通確認や，netperf や iperf を利用したノード間の帯域確認などが利用できる．この結果を TopologyManager（TM）に反映することで，実際のトポロジ情報を管理する．

ただし，実験によって取得されるべき情報は異なるため，実験実行者の実験設定に従って監視対象を決定する．また HC や SM がノードの障害を発見した場合は，DRM に通知し，使用停止などの処理を行う．

4.9 ログ収集

ログ収集は，LogManager（LM），LogCollector（LC）および LogAgent（LA）によって実現される．LogAgent はノード上で動作しログを残すアプリケーションであり，基本的には実験対象のアプリケーションが該当する．LC は，このようなアプリケーションが残したログを収集し保持する．LC は実験対象が動作するノードで動作する場合と，管理側のノードで動作する場合がある．実験対象のアプリケーションがログをファイルに直接書き出す場合が，LC が実験用

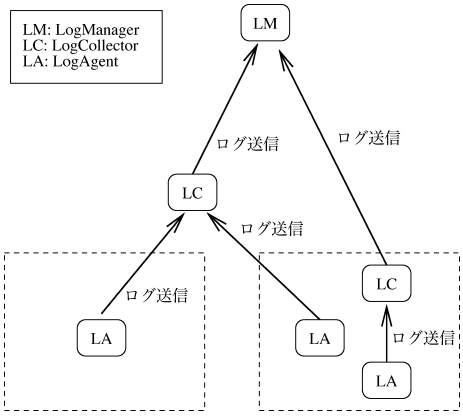

図 6 ログ収集の概念図
Fig. 6 Overview of log collecting.

ノードで動作する場合の一例である．この場合，実験対象のアプリケーションは実験ログを出力する LA とそれを保持する LC の 2 つの役割を持つ．Syslogd などを用いてアプリケーションのログを別のノードに送信し，そのノードに保持する場合が LC が管理側ノードで動作する例である．**図 6** にログ収集の概念図を示す．LM は LC が保持した情報を最終的に回収し，実験実行者が複数のノードにアクセスする手間を低減させる．また，ログの種類によっては，ログを解析しやすい形に整形するログ解析支援を行う．

このような 2 段階の制御を行うのは，各ノード上のログがそのノードに保存される場合と，別のノードに送信されて保存される場合双方に対応するためである．それぞれ，実験により向き不向きがあるため，双方に対応できる必要がある．

4.10 実証環境の初期化

実験の初期化は，対象のノードを利用した実験駆動単位を生成し，各ノードに標準の設定を行うシナリオを実行することで実現できる．

NodeCleanupManager（NCM）は DRM と連携し，予約割当て後などに，その実験で利用されたノード群の初期化を行う．手法はすでに述べたとおりだが，対象ノードを利用したノード初期化のためだけの実験を行う．

4.11 実験実行者による手動実験制御

ES はその時点で実行されている実験の EM の動作位置情報を保持しており，実験実行者は EM に対して，実験名や実験実行者名をキーに問合せを行うことで，EM へのアクセス方法を確認する．EM は，所属する実験駆動単位の制御モジュールの動作情報を保持しており，実験実行者から実験を遂行するためのモジュール群を隠蔽する役割を持つ．実験を制御するための様々なインタフェースを実験実行者に対して提供し，実験実行者からの要求に従い，そのほかのモジュールと協調することで実験を制御する．

4.12 資源利用の排他処理

資源利用の排他処理は様々なモジュールで対応が必要である．これまでに提案した各種モジュールでは，これについてはすでに共有される資源と，それに対する操作を行うモジュールで調停機構をおくこととしている．

4.13 実験駆動単位の制御

実験駆動単位が生成されると，その後は SCD が中心となって実験を制御する．実験全体をシナリオ実行と見なし，ノードの割当て要求や，ノードへのソフトウェア導入などもすべてシナリオの一部として実行するためである．これにより，実験中にノード割当て要求などを自由に行える．

4.14 実験状態の保存

実ノードを利用した実証環境では，ディスクのイメージをバイナリとして保存し，それを書き出すことで，ディスクの状態は保存できる．これらはノードへ導入するディスクイメージの作成と同様の手順で行える．しかし，ネットワーク状況やメモリ使用状態を保存することは困難である．

一方 VM Nebula では VMware[19] の機能を利用し，仮想機械のディスクイメージを保存することにより，ディスクの内容，プロセスの起動状態，メモリの内容などを保存できる．しかし，仮想機械はノードを模倣する機構であるため，ネットワーク上の情報は管理できない．

実験状態の保存については，実証環境の実現方法により，その可能性および実現方法が異なるといえる．

4.15 複数の実証環境の協調

ある実証環境の ES が設定記述を読み込み，設定記述に他の実証環境を利用するための記述があった場合には，他の実証環境の制御モジュールと協調した動作を行い，実験を行う．

設定ファイルが入力された実証環境の EM は，ローカルの実証環境を構築するとともに，他の実証環境上の ES に対して EM を起動する資源の割当てを要求し，EM を起動する．その後は，それぞれの EM が協調することで，複数の実証環境にまたがる実験駆動単位を形成する．各実証環境上の制御モジュールはそれぞれの実証環境上で動作する EM および SCD からの指示により動作する．各実証環境の EM および SCD どうしが協調することで，実験実行者からは複数の実証環境に資源が分散していることを意識することなく

実験が実行される．

4.16 例外処理

実験中の例外は基本的にはすべての制御モジュールおよび，実験用ノードで発生する．このような例外はEMに通知されることとし，実験実行者は実験設定に例外の種類とそれに対しての処理を記述する．例外に対する基本的な処理としては，実験の停止と障害を無視した実験継続が考えられ，実験の停止の場合は，実験実行者が次の指示を行うまで実験駆動単位をその状態で保存する場合と，自動的に初期状態に戻すなどの細かい処理も考えられる．

ノードやリンクの状態はSM，HCなどから，RIMおよびTMに通知されるため，これらの機構が不正状態を検知しEMに通知する．この通知を受けた，EMはSCDと協調することで対応する例外処理を行い，管理画面への表示や実験実行者へのメール送信などにより例外内容および，それに対する処理を通知する．また，実験実行者によりノードの故障などが検出された場合は，実証環境の管理者に障害情報を通知しなければならない．このような情報はDRMに送信され管理される．

実験駆動単位を構築する以前では，EM起動のための資源が確保できないなどの例外が考えられるが，このときはESから実験実行者に例外が通知される．

5. 考　　察

我々は，実証環境で実験を実行するにあたり，実験実行者が必要とする機能を整理し，それを満たす汎用アーキテクチャを設計した．

本章では，本アーキテクチャについていくつかの面から考察を行う．

5.1 汎用性

我々は，これまでのStarBEDとSpringOSおよびVM Nebulaの開発・運用経験と，実証環境やソフトウェアシミュレーションでの実験実行者からのインタビューから，実証環境への必要機能を2章にまとめ，この要求を満たすことのできる支援ソフトウェアのアーキテクチャの一例を提案した．インタビューの対象とした実験は複数の組織で行われた，まったく異なる目的のものであるため，多くの種類の実験には対応できると考えられる．また，3章で述べたように，ある特別な目的を持つ実証環境の機能を引き出すためには，専用の支援ソフトウェアが必要であるが，実験駆動単位を構築するための基本となる機能は実証環境により大きく変わることはない．したがって，本提案アーキテクチャを用いることで，実証環境の基本的な機能を引き出し，実験を行うことが可能である．

さらに，本節では一例として現SpringOSの機能と提案アーキテクチャで実現できる機能について比較する．

資源管理　SpringOSでは，静的資源および，割当て情報の管理を実現している．さらに，予約管理ができれば，柔軟に実験実験に対応できる．

設定読み込みおよび実験駆動単位の生成　SpringOSでは，実験実行者が手動で制御モジュールを起動させ，SCDに制御モジュールの動作ノードを指定する．実験制御に必要に性能を持っていないノードで制御モジュールを動作させてしまうおそれがあるため，実験駆動単位の制御モジュールを動作させるノードの管理・割当てが必要である．

ノード機能の指定および割当て　SpringOSではノードの機能はネットワークインタフェースの数および，種類でのみ行っている．実験トポロジの構築には十分な機能であるが，ノードの性能などを指定できることにより柔軟な実験駆動単位の構築が可能となる．

ノードの死活管理　SpringOSではWake on LANによる，ハードウェアレベルの電源投入および，SNMPによる電源停止，再起動処理をサポートしており，現時点である程度の制御は可能である．これに，他実験との調停機能を用意し，他実験へ割り当てられているノードへの操作の制限をできることが望ましい．

ノードへのソフトウェア導入および設定　SpringOSでは，本論文で提案した，PXEおよびTFTPを用いた起動方法変更により，ディスクレスシステムとしての起動および，ローカルディスクからの起動に対応している．またすでに紹介したこれら2手法の複合により，ディスクイメージの導入も可能である．

ノードでのコマンド実行　SpringOSでも，コマンド実行およびメッセージ交換によるノード間同期を実現している[20]．

トポロジの設定　VLAN設定によるトポロジの自動設定に対応している．さらに，他実験との調停機構を実装し，他実験に割り当てられているノードに影響を防げれば複数実験実行者環境に対応しやすい．

リンク特性設定　SpringOSではリンク特性は，各ノードでDummynetやNISTnetを起動するシナリオを実行することで実現している．実験設定記述が複雑になるという問題点があるが，支援ソ

フトウェアで対応し，実験実行者の負荷を低減させることができる．

実証環境および実験駆動単位の状態監視 実証環境から，SNMP や ICMP を利用して電源の入切を監視を行っている．NA と SCA に対応するプログラムの通信により，シナリオの実行状態を確認することは可能であるが，支援ソフトウェアでシナリオの実行状況などを確認する機構があることが望ましい．

ログ収集 ログの収集は，シナリオでログファイルを転送するなどという処理を実行することである程度対応できる．これについても，支援ソフトウェアで対応することで，実験実行者の負荷を軽減することができる．

実験環境の初期化 SpringOS では資源の予約管理を行っていない．実証環境の管理者などが手動で実行することで実現は可能であるが，ある程度自動的に実行されることが望ましい．

実験実行者による手動実験制御 実験実行者に対するインタフェースを特に用意していないため，実験実行者は ssh や telenet を用いてノードに接続し操作を行う必要がある．実験実行者が統一的に操作できるインタフェースを提供し，容易に実験を制御できれば実験実行のコストを低減させることができる．

例外処理 ノードの故障に備え，SCD に対応するモジュールが，余分にノードを要求する仕組みを持っている．ノード故障だけではなく様々な例外に対応できれば，より柔軟に実験に対応できる．

以上のように，SpringOS の機能は提案アーキテクチャですべて対応できる．また，VM Nebula では現状ノードの割当ておよびトポロジの構築のみをサポートしているが，この機能も提案アーキテクチャで対応可能である．SpringOS および VM Nebula はバッチ的な処理に対応していないが，提案アーキテクチャの ES の機能によりバッチ処理も可能である．また，SpringOS で現状では対応していない機能も，このアーキテクチャをもちいることにより実現可能である．

5.2 実証環境の協調

現在，Planetlab[21] や Netbed[22] では，様々な実証環境を独自の支援ソフトウェアで制御できるよう拡張が進められているが，実験設備および実証環境の一般化の議論は進められていない．実証環境に必要な機能を分離しモデル化することで，実証環境の協調が容易になる．単一の支援ソフトウェアを構築することは，すべての環境の機能を平均的に利用できるが，最大限活用することが難しい場合がある．特に VM Nebula のような環境では，実験用ノードは仮想機械であり，実ノード上で起動することが必要になる．したがって，実験支援ソフトウェアは，実ノードおよび仮想機械を階層的に管理する必要がある．このような特殊な環境をすべて単一の支援ソフトウェアで操作するのは困難である．提案アーキテクチャに従って実装されている支援ソフトウェアを利用すれば，支援ソフトウェアの実証環境で動作しているモジュールへアクセスできれば，内部を隠蔽することができ，これにより，柔軟な環境を構築できる．

DETER[23),24)] や SIOS，VM Nebula は，セキュリティ実験に特化した実証環境であり，StarBED や Netbed，Planetlab といった汎用的な実証環境とは異なる操作が必要となり，こういった実験施設上で動作する実験支援ソフトウェアは実験設備の性能を十分引き出すことができるように設計されるため，実験設備に機能を最大限引き出すことができる．汎用的な支援ソフトウェアを定義することにより，各実証環境の連携は容易になるが，前述のとおり各実証環境にはそれぞれ目的があり，その目的を満たすための実験設備が用意され，実験設備の機能を十分に引き出すための支援ソフトウェアが用意されている．汎用的なアーキテクチャを利用して各実証環境のすべての機能を引き出すことは難しいが，アプリケーションのインストールや，実験用トポロジの構築など基本的に実験を行うために必要な処理を透過的かつ自動的に行える可能性がある．また，複数の実証環境にまたがる実験駆動単位のノードの状態やトポロジの状況を，1 つのトポロジとして認識できる可能性がある．つまり，実験に必要なすべての操作を行うことは困難であるが，一部を自動化および一元管理することにより実験実行者の手間および時間的コストを軽減することが可能であると考えられる．

5.3 実証環境の評価基準

本論文で提案した一部の機能のみでも実験を実行することが可能であり，必ずしも，すべての実証環境が本論文で述べた機能すべてに対応する必要はない．したがって，ある実験が実現可能である実証環境とそうでない実証環境がある．たとえば，VM Nebula では前述のとおり実験用ノードの設定は実現しているが，実験シナリオの自動実行は実現していない．しかし，このような環境でも実証環境として十分な機能を持っているといえる．また，前述のとおり，現 SpringOS は，本論文であげたすべての機能を持っていないが，様々な実験に利用されてきた．ある実験の実現可能性

は，論文で述べた機能を評価軸とし，ある実証環境の持つ機能を検討することで測ることが可能となる．

6. おわりに

我々は StarBED と SpringOS，および VM Nebula の設計実装および運用を続けてきた．本論文では，これらの経験と，これまで様々な実験を行ってきた研究者にインタビューを繰り返すことにより，実証環境に必要な機能をまとめ，その要件を満たす汎用的なアーキテクチャを提案した．このアーキテクチャに従った実装を用いることで，実証環境の連携がより容易となり，柔軟な実験駆動単位の構築を実現できる．

すべての実証環境が持つ機能は一致している必要はなく，実験により使い分けられるべきである．このような場合に本論文で我々がまとめた機能を軸とし，実証環境を比較，選択することが可能となる．

実環境により近い環境で，容易に実験を行える基盤を提供することで，ネットワーク技術をより詳細に低コストで検証できる．また，実験実行のコストが低下するため様々なトポロジまたはパラメータを用いて，実験対象を様々な角度から検証できるため，実環境に与えるインパクトを事前に詳細に把握できる．今やコンピュータネットワークは社会基盤として利用されているため，コンピュータネットワークの障害は，様々なサービスを停止させてしまうおそれがある．我々が提案する基盤技術を利用すれば，詳細な実験を容易に行えるため，実環境での問題を減少させることができる．

参考文献

1) Miyachi, T., Chinen, K. and Shinoda, Y.: Automatic Configuration and Execution of Internet Experiments On An Actual Node-based Testbed, *International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities* (*Tridentcom*) (2005).
2) Miyachi, T., Chinen, K. and Shinoda, Y.: StarBED and SpringOS: Large-scale General Purpose Network Testbed and Supporting Software, *International Conference on Performance Evaluation Methodologies and Tools* (*Valuetools*) (2006).
3) 三輪信介，滝澤　修，大野浩之：仮想 PC インターネットセキュリティ実験環境『VM Nebula』の設計と構築，暗号と情報セキュリティシンポジウム（SCIS），電子情報通信学会 (2003).
4) 大野浩之，武智　洋，永島秀己：インターネットの脅威に対抗しうる脆弱性データベースと検証システムの構築，分散システム／インターネット運用技術（DSM）シンポジウム，情報処理学会 (2001).
5) 三輪信介，宮地利幸，大野浩之，篠田陽一：不正アクセス等再現実験環境の統合手法に関する研究 (2005).
6) 三輪信介，宮地利幸，大野浩之：不正アクセス等再現実験環境の統合実験，マルチメディア，分散，協調とモバイル（DICOMO2005）シンポジウム論文集，情報処理学会，pp.393–396 (2005).
7) IEEE standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks (1998).
8) Mills, D.: Network Time Protocol (NTP), RFC958 (1985).
9) Advanced Micro Devices, Inc.: *Magic Packet Technology* (1995).
10) Intel Corporation: *IPMI v2.0 specifications Document Revision 1.0* (2004).
11) Hewlett-Packard Development Company: *HP Integrated Lights-Out* (*iLO*) *Standard*. http://h18000.www1.hp.com/products/servers/management/ilo/index.html
12) Case, J., Fedor, M., Schoffstall, M. and Davin, J.: Simple Network Management Protocol (SNMP), RFC1157 (1990).
13) McCloghrie, K. and Kastenholz, F.: The Interfaces Group MIB, RFC2863 (2000).
14) Harrington, D., Presuhn, R. and Wijnen, B.: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks, RFC3411 (2002).
15) 三角　真，宮地利幸，知念賢一，篠田陽一：実ノードを利用したネットワークシミュレーションにおけるノードへの OS の導入及びパラメータ設定機構の開発，情報処理学会研究報告書 DPS-116, pp.95–100 (2004).
16) 宮地利幸，知念賢一，篠田陽一：StarBED における自動実験駆動機構，第 6 回インターネットテクノロジーワークショップ (WIT2004).
17) Rizzo, L.: Dummynet: A simple approach to the evaluation of network protocols, *ACM Computer Communication Review*, Vol.27, No.1, pp.31–41 (1997).
18) Group, N.I.T.: *NIST Net network emulation package*. http://www-x.antd.nist.gov/nistnet/
19) VMware. http://www.vmware.com/
20) Chinen, K., Miyachi, T. and Shinoda, Y.: A Rendezvous in Network Experiment—Case Study of Kuroyuri, *International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities* (*Tridentcom*) (2006).
21) Planetlab website. http://www.planet-lab.org/

22) White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C. and Joglekar, A.: An Integrated Experimental Environment for Distributed Systems and Networks, *USENIXASSOC.*, Boston, MA, pp.255–270 (2002).
23) DETER: DETER Laboratory for Security Research. http://www.isi.edu/deter/
24) Members of the DETER and EMIST Projects: CYBER DEFENCE TECHNOLOGY NETWORK AND EVALUATION — Creating an experiment infrastructure for develping next-generation information security technologies, *Comm. ACM*, Vol.47, No.3, pp.58–61 (2004).

(平成 18 年 7 月 6 日受付)
(平成 19 年 1 月 9 日採録)

宮地　利幸

2007 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．同年情報通信研究機構連携研究部門研究員．ネットワーク実験に関する研究に従事．博士（情報科学）．電子情報通信学会，日本ソフトウェア科学会各学生会員．

三輪　信介

1999 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．同年北陸先端科学技術大学院大学情報科学センター助手．2001 年通信総合研究所（現情報通信研究機構）研究員．インターネットのセキュリティおよびその再現実験環境の構築方法の研究開発に従事．博士（情報科学）．

知念　賢一（正会員）

1998 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．1998 年同研究科助手．2003 年北陸先端科学技術大学院大学情報科学研究科助手およびインターネット研究センター研究員．2006 年情報通信研究機構北陸リサーチセンター短期研究員．各種ネットワークサーバやネットワーク実験環境に関する研究開発に従事．博士（工学）．IEEE，電子情報通信学会，日本ソフトウェア科学会各会員．

篠田　陽一

1988 年東京工業大学大学院理工学研究科博士後期課程修了．1988 年同大学工学部助手．1991 年北陸先端科学技術大学院大学情報科学研究科助教授．2001 年同大学情報科学センター教授および同大学インターネット研究センタープロジェクトリーダー．2006 年情報通信研究機構北陸リサーチセンタープロジェクトリーダー．2006 年同機構情報通信セキュリティ研究センター長．情報環境，ネットワーク分散情報システム，ソフトウェア開発環境の研究に従事．工学博士．