修 士 論 文

ネットワーク実証実験環境のモジュール単位構築に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

村上 正太郎

2013 年 3 月

修 士 論 文

ネットワーク実証実験環境のモジュール単位構築に関する研究

指導教員　篠田陽一 教授

審査委員主査　篠田陽一 教授
審査委員　丹康雄 教授
審査委員　知念賢一 特任准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

1110060 村上 正太郎

提出年月: 2013 年 2 月

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　研究背景

インターネット上には様々なサービスが展開されており、インターネットは人々の生活に必要不可欠なものとなっている。また、インターネットは社会基盤として多数のユーザがサービスを利用している。新たなネットワーク技術のアイディアを検証する際や、ソフトウェアを実環境に導入する際に、動作に異常がないか、あるいは他のサービスに影響を与えないか等を検証する必要がある。検証方法として、実際に大規模なネットワーク実験環境を構築し、ソフトウェアの品質を向上させる実践的な検証が行われている。しかし、実験者が想定する大規模なネットワーク実験環境の構築には機材の費用・構築時間・人的コストが高く困難である。

大規模なネットワーク実験環境を構築するために必要な実験用ノードやネットワーク機器と、低コストで大規模な実験環境を構築可能な支援ソフトウェアが用意されたテストベッドが注目されている。その一つとしてインターネットから隔離された StarBED[1] がある。StarBED では、PC やネットワークなどの資源や設定、実行状態を管理し、ネットワーク実験環境を構築を容易に実行するための支援ソフトウェアとして SpringOS がある。StarBED ではノードの利用期間を区切り、実験者が自由にノードを設定できるように貸出している。そのため実験者は実験の検証を行う準備として、実験で扱うノードにソフトウェアのインストールから行なっていく。実験環境の構築に時間をとられてしまうと、StarBED での利用期間は決められているため、実験者は十分な検証を行う前に施設の利用期間が終わる。また、支援ソフトウェアの設定に想定以上の時間がかかることもある。

実験者は特定のレイヤ技術の検証が目的であるため、早々に環境構築を済ませ、検証段階へ移行したい要求がある。そこでこの要求を満たすため、記述の簡略化や実験環境の再利用を用いた少ない手順での実験環境構築が必要である。また、複数の実験環境が正しく動作することを確認後、大規模な実験環境として統合し実験を遂行することも考えられる。

1.2 研究目的

実験者が実験環境構築へ配慮する事項が少ないほど、ネットワーク実験の計画や実施が容易になる。ネットワーク実験環境における記述の構成を見直し、より簡潔な記述でネットワーク実験環境の構築を行うことを目的とする。

そこで本研究では、実験環境の一部を定義モジュールという単位で扱い、支援ソフトウェアを用いた実験環境の構築手法を提案する。これにより、実験者は目的に合わせて定義モジュールを組み合わせることで、容易にネットワーク環境構築の設定が行え、検証を開始することが可能となる。また、複数の実験者がそれぞれの実験環境の構築を行い、それらの実験環境を統合することで大規模なネットワーク実験環境とする。これにより、実験環境を並行して構築することが出来、実験環境の品質が向上し、実験環境の問題点の発見が容易となる。

1.3 論文の構成

2 章ではテストベッドと実験構築に関する既存研究について紹介する。3 章では、ネットワーク実験環境構築への実験者の要求を整理し、SpringOS の問題点を述べる。4 章では、容易な記述による実験環境構築手法について述べる。5 章では、実験環境の統合について述べる。6 章では、本研究のまとめと今後の課題について述べる。

第2章　関連研究

本章では、ネットワーク技術の検証のために用いられるテストベッドとして、PlanetLab[2]、Netbed[3]、StarBED について述べる。また、それらテストベッドの実験方法について述べる。

2.1　PlanetLab

PlanetLab は、米国の IT 企業や世界 60 以上の大学が中心となって 2003 年に立ち上げられたテストベッドである。PlanetLab の目的は、インターネットを利用した新しいアプリケーションやサービスに向けた、オープンで拡張性があるグローバルなネットワーク・テストベッドの構築することである。

PlanetLab のノードはさまざまな組織のネットワーク上に設置されている。広域に分散配置しているため、遅延やパケットロスといった、インターネットの特性を導入することが出来る。しかし、デメリットとして、問題発生時の原因の切り分けは難しくなっている。PlanetLab のノードは Linux の Virutual Server を利用しており、利用できる OS に制限がある。また、PlanetLab では実験遂行の機構は特に用意されておらず、利用者は手動か、シェルスクリプトや汎用スクリプト言語を用いて実験を制御する機構を各自で作成する必要がある。

2.2　Netbed/Emulab

Netbed は、分散システムと実ノードによる環境およびソフトウェアシミュレータの統合環境であり、豊富な実験管理機能を持つ。変更可能なパッチパネルをソフトウェアにより制御し、物理的な結線を変更しトポロジを変更でき、VLAN と組み合わせることで柔軟な実験トポロジを構築できる。ns-2 を実ネットワークに接続できるように拡張した nse を利用し，ソフトウェアシミュレータと実ノードによる環境を接続しており，ns-2 を利

用している部分では，前述の通り実環境と同一の実装は扱えない．ある程度の規模の PC クラスタを持つサイトを接続することで、大規模な環境を構築しており、実験トポロジがさまざまなサイトに分割されることがあるそのため実環境のリンク特性も導入できるが、サイト間の接続の問題が実験に影響をおよぼす可能性があるため、問題発生時の原因の切り分けは困難である。

Netbed では、実験者からのリクエストを受付け、設備が空き次第実験を実行する。実験は一括処理で遂行されるため、実験の状況を判断して次の内容を決める場合や、実験トポロジをある状態まで自動的に構築し、その後は実験者が手動により操作したい場合には不向きである。実行遂行は ns[4] がベースとしている Tcl 言語が支え、ns のシミュレーション全体と詳細を記述できる能力が、実験の全体と詳細を記述する能力に活用されている。

2.3 StarBED/SpringOS

StarBED は、情報通信研究機構北陸リサーチセンターによって提供されているテストベッドである。StarBED には 1000 台を超えるノードとそれぞれのノードを接続しているネットワーク機器から構成されている。StarBED のノードは、仕様に応じてグループと呼ばれる単位で区切られており、それぞれのグループのノードの性能は同一である。実験者は提供される論理ネットワークを構築するための VLAN 番号およびノードの一部を借りて実験を行う。ノードは実験者が任意に設定を変更可能であるが、全てのノードを OS 導入から手作業で操作することはコストが高い。そこで、StarBED ではネットワーク実験を支援するソフトウェアとして SpringOS[5][6] が存在する。SpringOS は様々なソフトウェア群の名称であり、ネットワーク実験の用途によって扱うソフトウェアが異なっている。SpringOS の主なソフトウェアについて述べる。

2.3.1 ERM : Experimemt Resrouce Manager

ERM は実験用の資源の管理・割り当てを行う。要求されるネットワークインターフェースの数と種類から適切な物理ノード、必要なネットワーク数の VLAN 番号を割り当てる。また ERM では各実験で提供されている以外の実験資源は利用できないようにし、一つのリソースを複数の実験に割り当てないよう状態も保存している。

2.3.2 ENCD : Experiment Node Configuration Driver

ENCD は実験実行の核となるものであり、実験者による設定ファイルから、他のモジュールに対して指示を送り実験を実行する。ENCD では、一連の役割により別々にモジュールが用意されている。たとえば、実験を実行するための ENCD は kuroyuri master として実装されている。また、ディスクイメージを作成する際の ENCD は pickup があり、導入する際の ENCD は wipeout が実装されている。pickup と wipeout は基本的には後述する NI と組として利用され、NI と通信することでその枠割を果たす。

2.3.3 NI : Node Initiator

実験者が実験で扱うノード構成は同じなことが多い。しかしネットワーク実験環境では、全てのノードへ OS のインストールを行うことは時間・労力とコストが高い。そこで、実験に使用するノード 1 台に OS やソフトウェアのインストールを行なっておくことで、支援ソフトウェアである NI[7] がノードのディスクイメージの作成後、同じ設定にしたい他のノードへ配布を行う。NI は各ノードのハードディスクに書き込みを行うため，専用のディスクレスシステム上で起動する。ネットワークブートからディスクレスシステムの OS を起動し、DMAN が NI が起動するように切り替える。NI 起動後，ENCD から指示されたディスクイメージをファイルサーバから取得し，ハードディスクに書き込みを行う。

2.3.4 pickup/wipeout

ディスクイメージは、SpringOS で提供しているプログラム pickup を用いて作成できる。実験実行者は実験用ノードに必要な OS やアプリケーションソフトウェアの導入、設定などを行い、ハードディスクのパーティションをイメージとして pickup で取得し、ファイルサーバに保存する。

ディスクイメージの書き込みのみ行いたい場合は wipeout で書き込む。実験の一部として OS 導入を行う場合は kuroyuri master(以下 kuma) によって書き込みが行われます。

2.3.5 DMAN : Directory Manipulator

実験用ノードはPXEでブートローダーを取得し、起動方法を決定している。各ノードのブートローダーに関する設定は、DHCPdの設定として記述されている。SpringOSではDHCPdの設定に記述するブートローダーのファイル名は固定し，そのファイルをシンボリックリンクとして，シンボリックリンクがさすリンク先のファイルを変更することで，変更の影響範囲を単一のノードのみとしている。DMANは、要求に応じてこのシンボリックリンクの設定を変更する。

2.3.6 Kuroyuri

Kuroyuri[8][9]は、ネットワーク実験記述言語処理系である。SpringOSでの実験遂行を担当し、その実験記述言語をKと呼ぶ。実験記述言語Kを用いて、実験全体の処理の流れと構成要素の挙動をまとめて実験シナリオとして記述する。実験シナリオは、管理用ノードで動作するkumaと、実験用ノードで動作するkuroyuri slave(以下kusa)が協調して実験を自動遂行可能とする。Kuroyuriによるマスタ・スレーブモデルの概念図を図2.1に示す。マスタが実行されるとスレーブにノードシナリオ等の部分スクリプトを転送し、スレーブはマスタから部分スクリプトを受け取ると即座に実行を行う。

スレーブ間は基本的に非同期であるが、kumaと実験用ノード上のkusaがメッセージを交換することで、利用者は必要に応じて全体の流れを制御できる。マスタ専用のシナリオを持っており、スレーブへのメッセージの受信するまでのシナリオの停止やメッセージの送信を行う。また、実験用ノードで実行されているシナリオ間の協調を行うこともできるこのモデルを図2.2に示す。

2.3.7 SWMG：SWitch ManaGer

StarBEDでは，様々なメーカのネットワーク機器がVLANを用いてL2レベルでセグメントを分離している。実験者はネットワーク実験環境構築の際、それらのネットワーク機器に対して設定する必要がある。しかし、スイッチはメーカによって操作方法が異なるため、スイッチ制御の違いに時間を費やすことになる。そこでSWMGは様々なネットワーク機器の設定方法の違いを吸収することで、実験者は同じ使い方でネットワーク機器

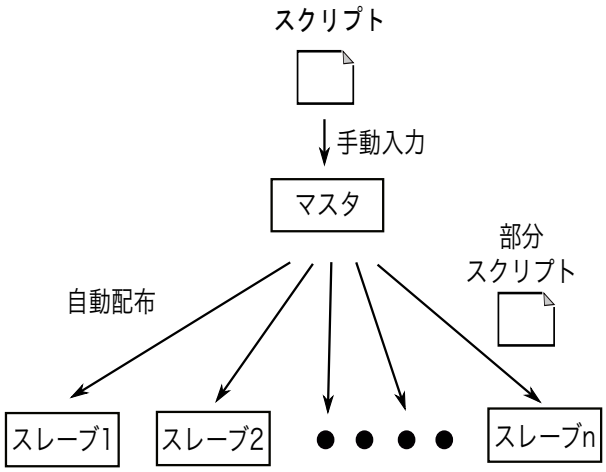

図 2.1: マスタ・スレイブモデル

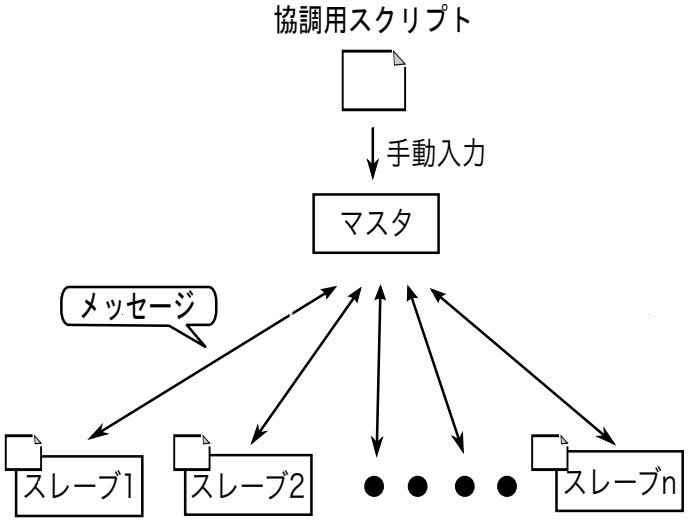

図 2.2: マスタ・スレイブ間のメッセージ交換

にVLANの設定を反映することが可能となる。また、ERMと強調することで実験者がスイッチポートやVLANの設定が変更可能か確認を行っている。

第3章　ネットワーク実験環境構築における要求分析

本章では、実験者のネットワーク実験への要求を整理し、それらの要求に対応する既存研究について述べる。次にネットワーク実験を支援するソフトウェアである SpringOS の実験手順を示し、SpringOS を用いたネットワーク実験の現状を整理する。最後にネットワーク実験環境における記述の要求について述べる。

3.1　ネットワーク実験環境への実験者の要求

本節では、実験者がネットワーク実験環境に求める機能を整理する。これらの機能は基本的にすべての実験環境に要求される機能である。

1. 実験の容易性
   実験の準備にかかる時間と実験の実行にかかる時間が短いと、検証の解析に時間を長く利用できる。

2. 段階的な構築
   実験環境を一括で構築し、実験を行うことは難しい。環境構築を順序立てて行うことで、問題が発生時の問題点の切り分けが容易になる。

3. 支援ソフトウェアの利便性
   ネットワーク実験を支援するソフトウェアが使いやすいほど、実験の遂行が容易になる。特に以下の項目を満たすことで実験者は容易に実験の遂行が可能となる。

   (a) 自動的な実験遂行

   (b) 実験トポロジの容易な構築

   (c) 実験ノードの自動設定

4. 検証結果の精度向上
   得られる実験データの結果の精度が良いほど、実験の評価が行い易くなる。

5. 実験状態の保存
   施設によっては利用できる期間が限られているため、期間内に必要な実験を終了できない場合、実験を一時中断し後日実験を再開する必要がある。

6. 実験環境の再利用
   他の実験者が構築した環境を利用することで、実験者が全ての環境構築を行わずにすむ。

3.2 実験遂行における既存研究

実験遂行とは実験の環境構築から実験の検証までを含めたこことである。下記では、実験で扱う資源の効率的な利用方法とノードの準備に関する研究と、検証精度の向上と実験で扱うソフトウェアの利便性に関する研究について述べる。

環境構築段階に関する研究

実験環境を構築する際、１台の物理ノードを多重化することで、実験環境の拡大が可能となり、物理ノードの効率的な利用が可能となる [10]。また、実験資源に関する管理システムをカスタマイズすることで、実験資源の制限の少ないネットワーク実験設備として扱うことが可能である [11]。実験環境の保存・復元の仕組みも提案されており、実験者が再度実験を行う場合、作業の削減を可能とし、実験を円滑に行える [12]。

実験準備の所要時間を短くするため、複数ノードへのイメージの配布の高速化に関する研究も行われている。FAI(Fully Automaition Install) を利用したノードへの OS イメージ投入の高速化におけるインストール手法の開発され、イメージの作成と配布の高速化も行われている。

検証段階に関する研究

検証結果の精度を上げる研究として、遅延やパケットロスといったインターネット特性を模倣したネットワーク実験環境の構築に関する研究 [13] や、一般的な通信ソフトウェア

を起動することでバックグラウンドトラフィックを発生させる研究がある [14][15]。

ソフトウェアの利便性を向上させる研究として、ノードの起動や停止の流れ、つまりノードの活動サイクルを形作る要素技術をインタラクティブに操作する研究 [16] や、SpringOS の各 API を様々なスクリプト言語で利用可能にする研究がある [17]。

3.3 SpringOS の現状

SpringOS を用いた実験環境の遂行の流れを述べ、実験者にとって SpringOS が有効的に扱える実験と現状について述べる。

3.3.1 SpringOS を用いた実験の手順

既存研究である SpringOS に注目し、SpringOS を用いた実験の手順を述べる。SpringOS は実験環境の構築からシナリオ実行までの実験遂行を支援しており、実験結果に関する検証までは含まれない。

1. 実験の検討
   問題点や課題を明確し、実験の目的を達成するための適切なネットワークトポロジや各ノードへ導入するソフトウェアを検討し、実行するシナリオの検討をする。

2. 汎用ノードの準備
   ノードへ汎用的に扱う OS とアプリケーションソフトウェアを導入する。pickup が設定を行ったハードディスクのパーティションをイメージとして作成し、ファイルサーバに保存する。

3. 実験シナリオの記述
   検討の結果にしたがって、実験者はトポロジ構成やノード構成の設定と、それらを用いた実験のシナリオを記述する。

4. SpringOS の実行

   (a) 実験シナリオの読み込み
   kuma/kusa の動作を制御するための設定ファイルが必要となる。管理ノードで

作動する kuma は設定ファイルと実験設定記述を解析し、実験者の要求を認識する。

(b) 実験資源の割り当て
kuma は構築するために必要なノードおよび、VLAN ID を ERM に要求する。ERM は要求された資源を検索し、用意できれば割り当てる。

(c) ノードへのディスクイメージの導入
wipeout は設定記述にしたがい、ディスクイメージを取得し、ハードディスクに書き込みを行う。

(d) 実験用スイッチの設定
実験記述に従い SWMG を通じて実験トポロジを構築する。

(e) 実験シナリオの実行
実験で用いるノードでは kuroyuri slave(以下 kusa) が起動されている必要がある。kusa が起動をすると kuma からの通信を待ち受け、kuma から受け取ったシナリオを実行し、実験データを収集する。

5. 実験結果の解析
実験前に上げた問題点・課題が実験のデータを解析することで正当性を保証することが可能か確認する。

3.3.2 SpringOS を用いたネットワーク実験

実験者は実験シナリオを記述し、SpringOS に読み込ませることで、実験シナリオに沿った環境構築とシナリオを実行する。実験シナリオはノード単位で記述され、ノードに満たすべきノード仕様とノードシナリオが記述される。ネットワーク実験では多数のノードを扱うことが多いため、参考となるノードをノードクラスとして定義することで、同じノード仕様とノードシナリオを持つノードを一度に多数生成することが出来る。ノードクラスからノードを複数生成することで、実験者は容易に大規模な実験環境を記述することが可能である。しかし、現在の SpringOS ではノード仕様かノードシナリオのどちらかが異なれば新しくノードの記述を行わなければならない。様々な仕様とシナリオをもつノードから構成される複雑なネットワーク実験環境の記述には、実験環境の規模に比例して記述量は増加する傾向にある。

実験環境はノード単位で表現され、ノードの仕様として下位レイヤか記述する。そのため、下位レイヤに興味のない実験者や上位レイヤの実験を行う実験者も下位レイヤから実験環境全体を記述する必要があり、実験者は実験対象を含めた実験環境の詳細を全て記述する必要がある。これは実験環境を構築する際の簡便性の面で問題がある。

実験者は実験シナリオファイルに実験環境とシナリオを記述する。実験シナリオファイルは複数に分割可能であるが、kuma は入力された実験シナリオファイルから評価するため読み込ませる順序には気をつけなければならない。この例を図 3.1 に示す。この方法では他の実験者が作った実験シナリオファイルを template としてそのまま扱うことが出来る。しかし実験者が template の実験環境の仕様変更をする際には、ファイルの中身を書き換える必要がある。そのため、仕様変更に対して柔軟性が高いとは言えず、再利用性も高いとは言えない。

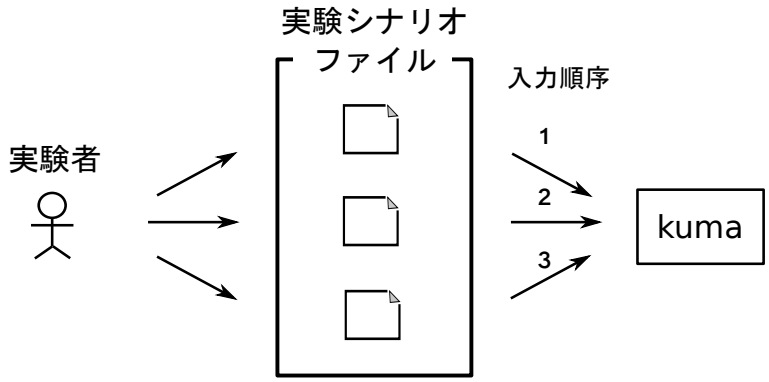

図 3.1: 実験シナリオの分割

3.4 ネットワーク実験での記述への要求

実験記述の構成について検討をする。大規模実験での実験記述に求められる項目を述べる。

- 実験のカプセル化
  実験環境とその振る舞いをグループ化として１つの抽象物と扱う。実験者が利用する場合、細かい仕様や構造は隠蔽化される。実験者は、内部構造や動作原理の詳細を知る必要はなく、決められた手段を用いることで、決まった振る舞いをする実験

環境を構築することが可能となる。特に大規模な実験記述では、可読性や開発効率の向上が期待できる。

- コードの共有のよる再利用性の向上
  独立性が高いと、それだけを切り出して使うことが容易である。ライブラリを利用することで、実験者は他の部分の作業により専念でき、開発効率が高まる。プログラムの部分的な開発・再利用が容易になる。

- 記述必要項目の最小化
  実験に伴って配慮する事項が少ないほど、ネットワーク実験の計画・実施が容易になり、実験の作業効率が向上する。実験者が気にしなくても良い項目を補うことで、実験者は意識を実験内容に集中できる。

これらの構成を取り入れた実験記述の提案を行う。これにより、実験者は複雑なネットワーク実験環境の表現が容易になり、実験対象に注目した実験環境の構築が可能となる。

第4章　定義モジュールを用いた実験環境作成

設計するシステムの全体像・構成を 4.1 節で述べる。4.2 節で定義モジュールに必要な要件を述べ、4.3 節で設計を行う。定義モジュールを扱う M 言語の仕様を 4.4 節で述べる。4.5 節でシステムの構成について述べ、4.6 節で実装を述べる。4.7 節では

4.1　全体像・構成

提案する実験ネットワークを構築するシステムの概要を図 4.1 に示す。本システムでは、実験ネットワーク構築のために、ユーザが記述を行う実験シナリオファイルと実験環境をライブラリ化しているファイルを用いて実験環境を表現する。この実験環境をモデル化したライブラリファイルを定義モジュールと呼ぶ。定義モジュールを実験記述ファイルからのみ利用することで、定義モジュールに何らかの変更を行なっても、他のライブラリに与える影響は少ない。また、定義モジュールはクラスと同様に扱うため、定義モジュール単体でネットワーク実験環境の構築は出来ない。定義モジュールをパラメータを用いてインスタンスすることで元ファイルに変更を加えずに環境構築が可能となる。提案する実験環境の構成は、StarBED で用いられている SpringOS を基に設計を行い実装した。

4.2　定義モジュールへの要件

実験環境とその挙動をひとまとまりの単位として複合化したものを定義モジュール (Defined Module)、以下 Dodule と定義する。実験環境のモデル化・機能拡張や仕様変更の容易さ・可読性の向上から、Dodule はオブジェクト指向的なクラスの概念を導入している。本章では Dodule に求める要件を述べる。

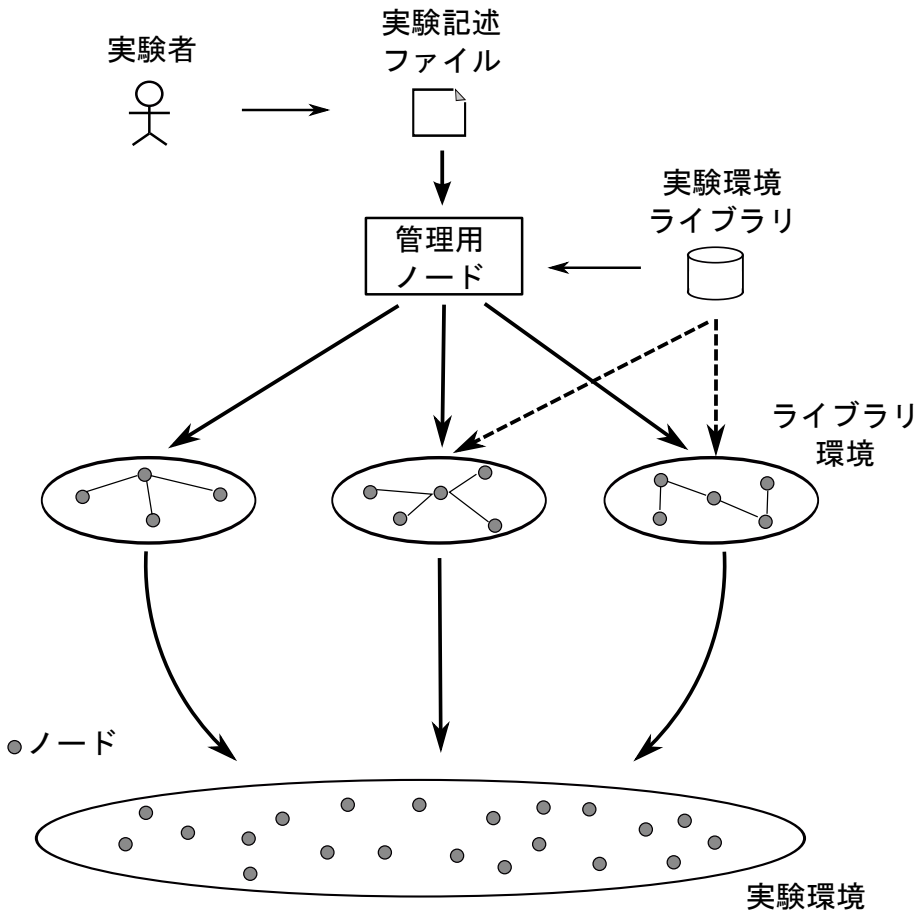

図 4.1: 全体の構成

1. 実験環境のモデル化

   ノードやネットワークなどの構成要素の集合や、実験環境といった大きな要素の挙動を抽象化する。必要とする部分のみに注目して実験環境の特性を抽出し、Dodule として定義する。これにより、実験者は Dodule の内部構造や挙動の詳細な設定は知る必要はなく、パラメータを与えることで実験者の想定する環境を構築することが出来る。特に大規模ネットワークを構築する際、実験者は Dodule を利用することで実験環境の全てを記述する必要がなくなり、より少ない記述で想定する実験環境を構築することが可能である。

2. パラメータライズ

   一般的に、実験はさまざまなパラメータを変更しながら繰り返されることが多い。そこで、実験パラメータの変更が容易であれば実験実行にかかるコストが小さくな

る。パラメータライズのメリットとして、汎用的な環境に実験者の任意のパラーメタを付与可能とすると別の実験などでの再利用が容易となる。

3. 柔軟な実験の記述
   実験者が機能を追加した新しい Dodule を再定義可能とする。Dodule を再利用することで新たに全てを作るという手間を削減でき、既存のテスト済みの要素の使用による実験環境の信頼の向上も期待できる。

4. 多態性
   同じ Dodule を扱う際、インスタンス毎でその性質が変更可能とする。それにより、一つの Dodule で異なるノード仕様やノードシナリオが容易に構築することが可能となる。

4.3 設計

前章で受けた要件を満たす Dodule を設計する。Dodule は実験で扱うノードとその挙動を定義したクラスであるため、Dodule だけでは実験環境を構築することは出来ない。そのため、実験者が Dodule を用いて実験環境を構築するにはインスタンス化、つまり具体化する必要がある。Dodule はノードとネットワークと Dodule からなる 3 つのメンバーで構成が可能である。Dodule の扱いの概念を図 4.2 に示す。図 4.2 で扱う Dodule は基本的な機能をもつ NAT とその NAT の内側に複数台のノードが参加している Stanndard_NAT Dodule である。この Standard_NAT Dodule は仮引数として NAT の内側のノード数を持つ。そのため、Standard_NAT Dodule を利用する際は、必要な実引数を与えることでインスタンス化することができる。Dodule の特徴的な機能について各章で説明をする。

4.3.1 柔軟な引数

引数として、整数、文字列、ネットワーク、ノードといった実体からノードクラスや Dodule などのクラスを扱うことが可能である。Dodule で扱うメンバーの定義だけし、それらの挙動やどのように実装するかは実験者に任せることが可能になる。図 4.3 の Standard_NAT Dodule の例では、NAT の内側のノードは実験者が定義を行う。そのため、NAT の内側のノードにはノードしての仕様・役割は Standardu_NAT Dodule では定義されな

い。インスタンス化する際に利用者がノードの仕様・役割を実装する必要がある。図 4.3 ではノードの仕様として CentOS がインストールされているノードとそれらが 3 台に実装される引数を与えている。

4.3.2 コンストラクタ

Dodule に複数のコンストラクタを用意してくおくことで、Dodule の初期値が異なるインスタンス化を行うことが可能となる。コンストラクタを用いた Dodule の Standard_NAT の概要を図 4.4 に示す。Standard_NAT モジュールを用いてインスタンス化する際、NAT の内側のノードをデフォルト値かユーザ指定か選ぶことができる。図 4.4 の①では引数を 1 つで、Standard_NAT Dodule のデフォルトの初期を用いてインスタンス化が行われる。②では PC_num の値以外に、CentOS がインストールされているノードの指定を行うことで、ユーザ任意の NAT の内側ノードのインスタンス化が可能となる。

4.3.3 Dodule の組み合わせ

ノードとネットワークと Dodule を組み合わせ、新しい Dodule として再定義が可能である。既存 Dodule を利用することで、不具合を作り込む可能性も減らすことができ、実験者同士での再利用が行い易くなる。図 4.5 では Standard_NAT を用いて多段 NAT を行う Multigate_NAT Dodule を定義している。

4.3.4 オーバーライド

Dodule のメンバー要素とその属性をオーバーライド可能とする。ユーザが Dodule を扱う際、パラメータとして扱っていないメンバーの変更も可能とするために必要である。インスタンス化前に Dodule のメンバーのオーバライドを行った際、記述ファイル内では再定義後は再定義前と同じ Dodule は利用できない。

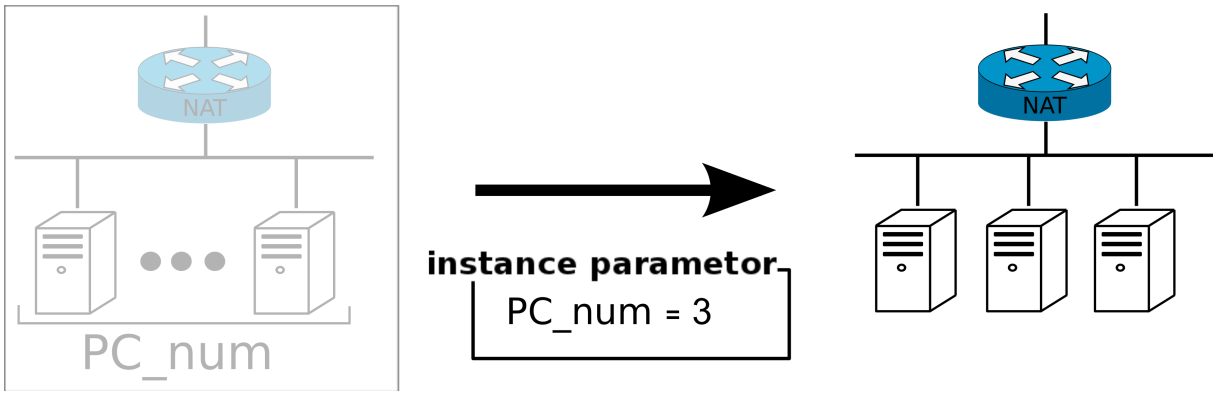

図 4.2: Dodule の扱い方

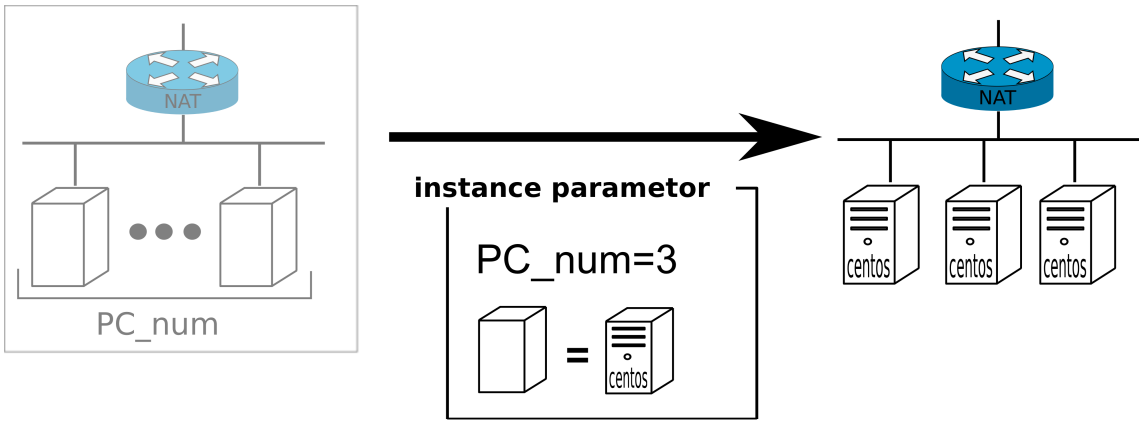

図 4.3: Dodule のパラメータ

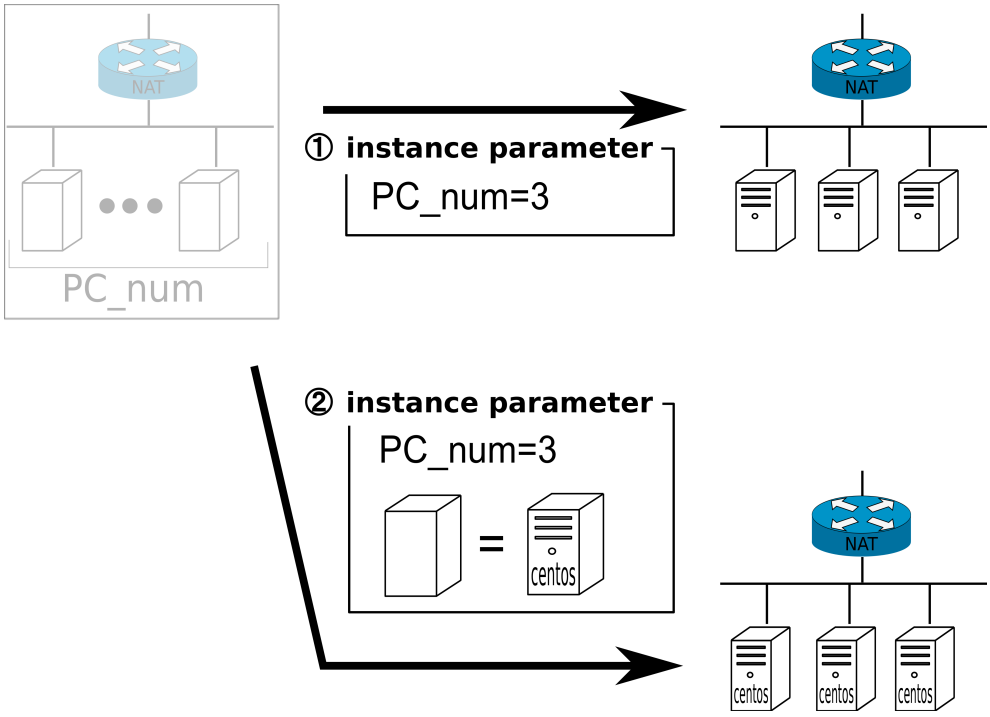

図 4.4: コンストラクタによる初期値

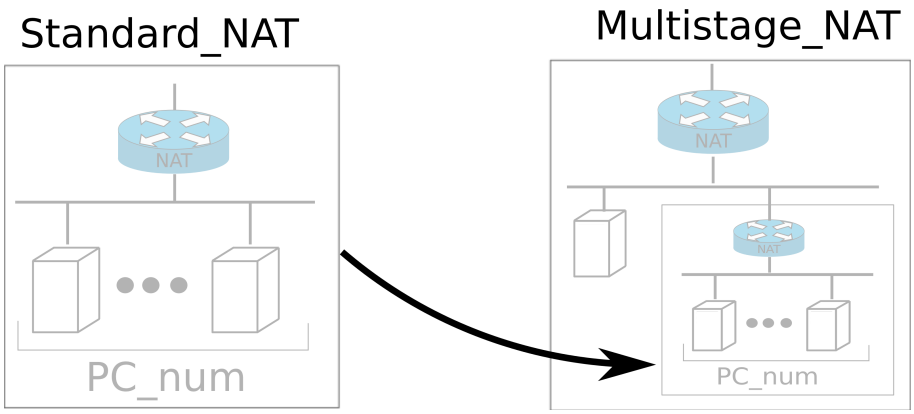

図 4.5: Dodule の組み合わせ

4.4 M 言語仕様

前節の要件を満たす言語処理系の設計を述べる。Dodule を扱える記述言語を M と呼ぶ。M は SpringOS のネットワーク実験記述言語である K を基に設計した。K 言語をラッパーしており、K 言語で扱える機能は利用可能である。

M 言語でのメインスクリプトを図 4.6 に示す。制御対象の挙動を記述する実行文をシナリオと呼び、ノードの挙動と全体の挙動を記述したシナリオをそれぞれ、「ノードシナリオ」「グローバルシナリオ」と呼ぶ。グローバルシナリオは基本的にノード間の同期に使われる。M 言語では実験全体で共通に用いられる変数を記述したと使用する Dodule の利用宣言後、ノード記述、ネットワーク記述、Dodule 記述、グローバルシナリオをスクリプトに記述する。それぞれの記述説明は以降の節で行う。

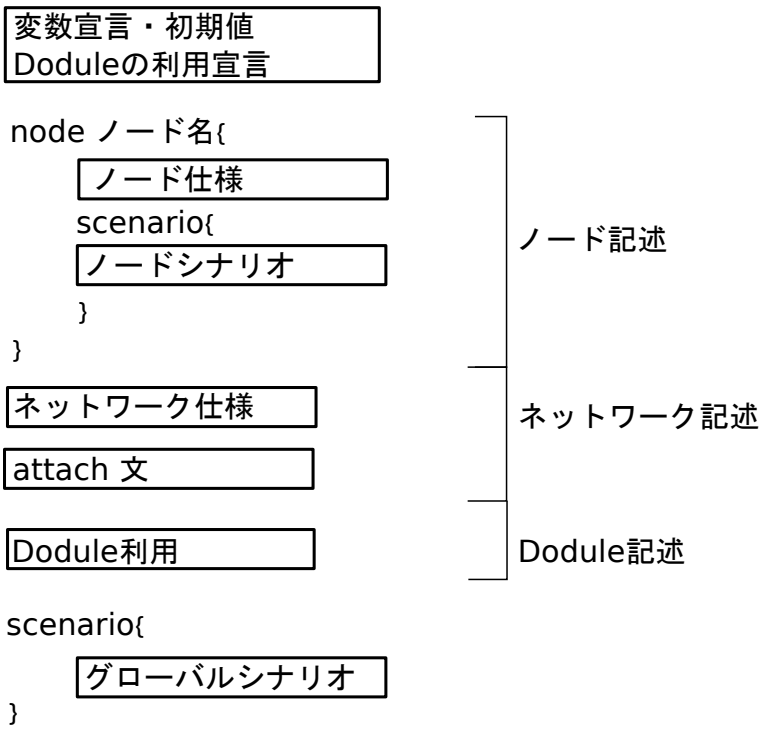

図 4.6: スクリプトの骨組み

4.4.1 データ型・変数・Dodule利用宣言

ネットワーク実験の記述では同じデータが繰り返し現れるので、変数宣言・初期値で他のプログラミング言語と同様に、変数を扱うことが可能である。実験全体で共通に用いられる変数を記述することで、記述の繰り返しを減らすことが可能である。このデータ型や変数の扱いはKと同じである。典型的な四則演算、変数のインクリメントとデクリメント等の演算子を持つ。変数の型は動的にきまり、型の宣言は不要である。グローバルシナリオとノードシナリオでは名前空間が異なるため、それぞれに含まれる変数は独立している。そこで、グローバルシナリオからノードシナリオへ変数を継承する機能がある。これにより、双方で同じ名前の変数が利用可能である。継承する変数の指定はexportを用いる。変数の値は継承時点でのみ同じ値を持つため、それ以外の箇所で値は一致しない。次の例ではノードAで変数fooの値31が出力される。スクリプトは上から評価されるため、ノードAで70は出力されない。また、バイナリファイルが置いてあるディレクトリをbindirとし、さらにechoコマンドのパスをpathechoと定義することができる。

```
foo = 31
export foo

bindir = "/bin/"
pathecho = bindir + "echo"
export bindir
export pathecho

node A {
    ...
    scenario {
        print foo
        pathecho hoge
    }
}
scenario{
    foo = 70
}
```

M言語ではDoduleの利用宣言を明示的に行うようにした。これは実験者に名前空間の制限を与えないためであり、不要な予約後を作らないためである。そのため、実験者は利用するDoduleの利用宣言以降は、宣言したDoduleのみ利用可能である。例えばStandard_NAT Doduleの利用を宣言する際には、

```
import   Standard_NAT
```

と記述することで、Standard_NAT Dodule が利用可能となり、Standard_NAT は予約後として扱われる。

4.4.2 ノード記述

実験設備の詳細な把握や記述するのは難しいため、ホスト名、IP アドレス等でノードを指定をせず、ノードが満たすべき仕様とシナリオを記述する。これにより、実験者は実験施設の条件を把握した記述の必要がなくなる。ノードのブート方法や、ディスクを利用する場合はディスクイメージの指定、ディスクイメージを書込むハードディスクのパーティション、必要なネットワークインターフェースの種類、ノードで実行されるシナリオなどの属性を記述する。netif はネットワークインターフェースでメディアが GigagitEthernet であることを示している。ここでは、2 つのギガビットイーサネットのインターフェースを持つノードが 1 つ生成される。**scenario** ブロックには、ノードの役割としてのシナリオを記述する。この例では、文字列 hello が出力される。

```
node A {
    netif[2] media gigabitehter
    scenario {
        print "hello"
    }
}
```

参考となるノードをノードクラスと定義することで、多数のノードを生成が可能となる。**nodeclass** ブロックでは、**node** ブロックと同様にノード仕様とノードシナリオの記述を行うが、ノードの割り当ては行われない。下記の例では、B という名前のノードクラスを定義している。ノードクラス B は、ネットワークインターフェースを 2 つ持ち、touch コマンドを用いて/tmp/huga ファイルを作成する。**scenario** ブロックでの **callw** 文は外部プログラムの終了を待つが、**call** は終了を待たず実行する。

```
nodeclass B {
    netif[2] media gigabitether
    scenario {
        callw "/usr/bin/touch" "/tmp/huga"
    }
}
```

ノードクラスからインスタンスを作成するには、**new** を用いる。次の例では先ほど定義したノードクラス B のインスタンスを作成している。1 行目では 1 つ生成され、2 行目では 10 つ生成される。ノードクラスから複数作成された集合を「ノード集合」と呼ぶ。

```
nodeb = new B
nodeb[10] = new B
```

4.4.3 ネットワーク記述

実験ネットワークは **network** で作成し、作成したネットワークに対して **attach** 文を用いてノードがネットワークに参加する。**attach** 文はノードのネットワークインターフェースと参加するネットワークを指定する。次の例では、2 つのネットワークに先ほどのノードを接続する。**ipaddrrange** 文はネットワークに IP アドレスの範囲を指定する。ネットワーク X には、ノード A の第 1 ネットワークインターフェースとノード集合 B の第 1 インターフェースが参加している。ネットワーク Y には、ノード集合 B の第 2 インターフェースが参加している。図 4.7 に実験ネットワーク群の構成を示す。

```
network X,Y
X.ipaddrrange = "192.168.0.0/24"
Y.ipaddrrange = "192.168.1.0/24"
attach A.netif[0] X
attach B[0-2].netif[0] X
attach B[0-2].netif[1] X
attach B[3].netif[0-1] Y
```

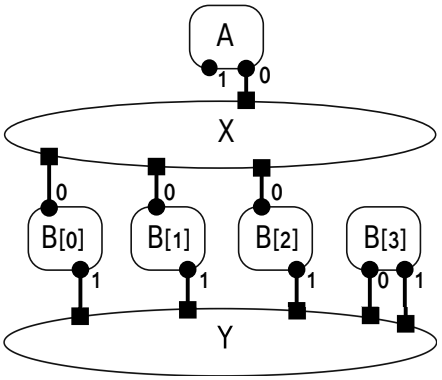

図 4.7: ネットワーク記述と構成

4.4.4 シナリオ記述

シナリオ記述の主な仕様を紹介する。それ以外の詳しい仕様はKのリファレンスを参照。

インターフェース属性取り込み

OSやハードウェア構成に応じてネットワークインタージェス名は異なる。そこで、デバイス名等のインターフェース属性を取り込む**netiffit**文がある。次のようにすることで第一インターフェースにIPアドレスを設定できる。

```
netiffit "/bin/ifscan"
...
callw "/usr/sbin/ifconfig" "self.netif[0].rname" "self.netif[0].ipaddr"
```

netiffit文は**ifscan**で得たインターフェースの属性をノード自身を示す特別な変数**self**に格納する。**self.netif[0].rname**は第一インターフェースのデバイス名、**self.netif[0].ipaddr**は同インターフェースのIPアドレスを示す。

メッセージ交換と同期

ノードシナリオとグローバルシナリオではメッセージの送信は**send**文を用い、ノード集合の全要素にメッセージの送信を行うには**multisend**文が用いられる。それらのメッセージを受け取るのは**recv**文が用いられる。メッセージ送信だけでなく、メッセージ交換することでノード間での協調が可能となる。受信したメッセージによって処理を切り替えたり、特定メッセージが届くまでの待機などが可能となる。メッセージを受け取るまで待機する**sync**文と、指定されたノードから届いたメッセージと文字列比較する**msgmatch**文がある。ノード集合の全要素からの文字列比較には**multimsgmatch**文を用いる。次の例では、グローバルシナリオが”START”メッセージをノードAとノード集合Bに送り、各ノードは受けっとたメッセージに従った処理の分岐をし、”START”メッセージの場合グローバルシナリオに”OK”メッセージを送る。グローバルシナリオは全てのノードからノードAとノード集合Bか”OK”メッセージを受け取とると、”experiment end”と表示する。

グローバルシナリオ

```
send A "START"
multisend B "START"
sync {
    msgmatch A "OK"
    multimsgmatch B "OK"
}
callw "/bin/echo" "experiment end"
```

ノードシナリオ

```
msgswitch x{
    "START"{
        ...
        send "OK"
    }
    "STOP"{
        ...
    }
}
```

4.4.5 Dodule 記述

Dodule スクリプトはメインスクリプトとは別ファイルに作成し、1 つのファイルに 1 つの Dodule を記述する。Dodule にはメインスクリプト同様に変数宣言、ノード記述やネットワーク記述、Dodule 記述を記述する。基本的な Dodule の骨組みを図 4.8 に示す。メインスクリプトと異なる点はグローバルシナリオが記述できないことである。Dodule 毎のグローバルシナリオは実験全体のシナリオと競合する可能性があるため、グローバルシナリオはメインスクリプトのみ記述可能とした。

NAT に関する例を基に 4.3 節の設計した特徴的な機能の扱い方を述べる。図 4.9 の Standard_NAT Dodule は、メンバー要素として以下の 3 つを持つ。

- ネットワークインターフェースを 2 つ持つ NAT の挙動をするノード
- NAT の内側のネットワーク
- NAT の内側のネットワークに参加しているノード

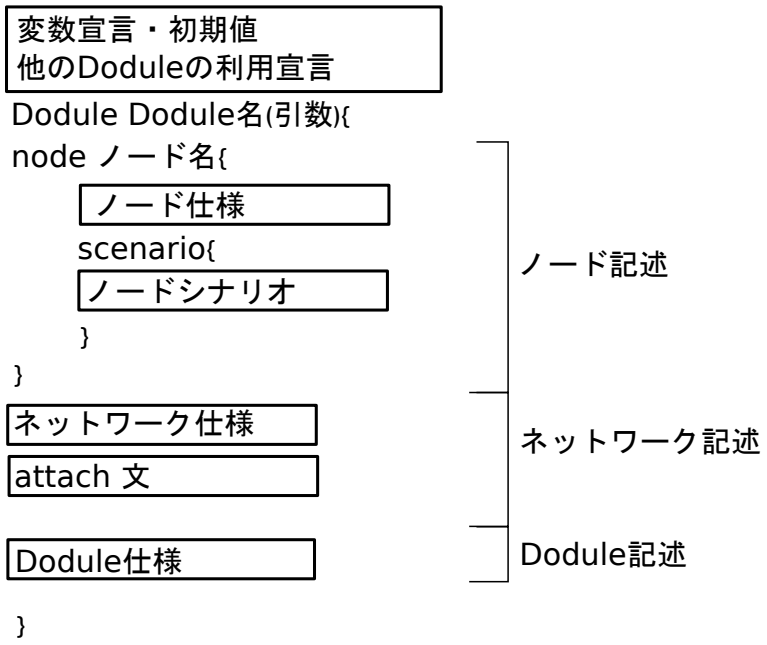

図 4.8: Dodule script

1 行目でインスタンス化する際に必要なパラメータの指定を行なっている。仮引数として NAT の外部ネットワークを exn とし、内部ネットワーク参加するノードの数を inode_num としている。2,3 行目は NAT の内部ネットワーク inn の作成とアドレス空間の属性を追加している。5 行目から 14 行目まではネットワークインターフェースを 2 つ持つ基本的な NAT の動きをする SNAT ノードを生成している。16 行目で引数と受け取っているネットワーク exn に SNAT の第 1 ネットワークインターフェースが参加し、17 行目で先ほど作成したネットワーク inn に第 2 インターフェースが参加している。19 行目から 27 行目では NAT の内側のネットワーク inn に参加しているネットワークインターフェースを 1 つ持つ INODE ノードクラスを定義し、引数の inode_num だけ生成する。

メインスクリプトで Standard_NAT Dodule を利用するために、**import** 文を用いて Standard_NAT を読み込む。NAT の外部ネットワークとして external_network を作成し、属性として 192.168.0.0/24 のアドレス空間を追加している。インスタンス化する際、仮引数と実引数の順序は対応しているため同じにする必要がある。そのため第一実引数と第二実引数にはそれぞれ、先ほど作成した external_network ネットワークと作成したい内部ネットワークのノードの数を指定している。実引数を与えてインスタンスした例は図 4.10 である。

```
1: module Standard_NAT(exn,inode_num){
2:   network inn
3:   inn.ipaddrrange = "192.168.0.0/24"
4:
5:   node SNAT{
6:     netif[2] media gigabitethernet
7:     scenario{
8:       netiffit pathifscan
9:       callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
10:      callw "/sbin/ifconfig" self.netif[1].rname self.netif[1].ipaddr
11:      callw "/sbin/iptables" "-t" "nat" "-A" "POSTROUTING"
12:                  "-o" self.netif[0].rname "-j" "MASQUERADE"
13:      callw "/sbin/service" "iptables" "save"
14:    }
15:  }
16:  attach snat[0].netif[0] exn
17:  attach snat[0].netif[1] inn
18:
19:  nodeclass INODE{
20:    netif media gigabitethernet
21:    scenario{
22:      netiffit pathifscan
23:      callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
24:    }
25:  }
26:  inode[inode_num] = new INODE
27:  attach inode[0-inode_num].netif[0] inn
28:
29:}
```

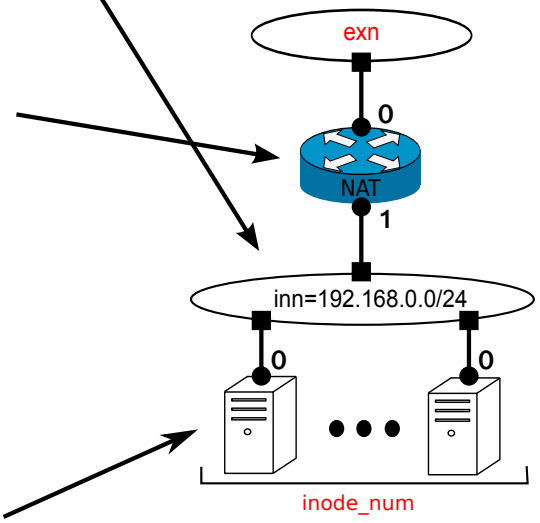

図 4.9: Standard_NAT dodule

メインスクリプト

```
import Standard_NAT
//NAT の外部ネットワーク
network external_network
external_network.ipaddrrange = "192.168.100.0/24"
//Standard_NAT Dodule のインスタンス化
nat_network = new Standard_NAT(extearnl_network, 3)
```

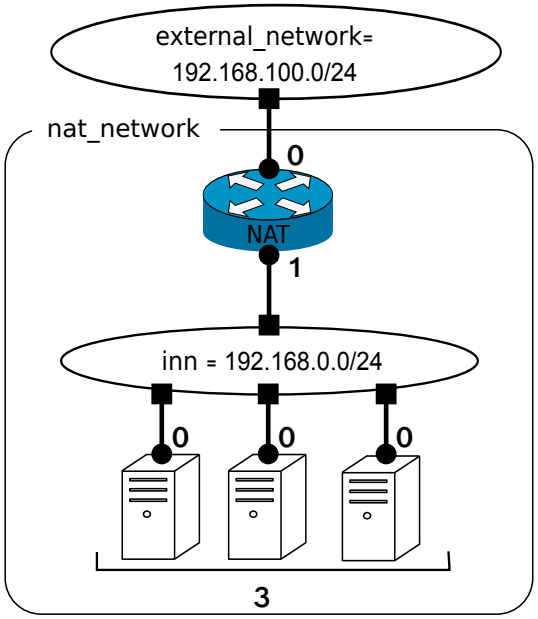

図 4.10: Standard_NAT のインスタンス化の例

引数

Dodule には複数の引数を指定することができる。引数として、整数、文字列、ネットワーク、ノードといった実体からノードクラスや Dodule などのクラスを扱うことが可能である。Dodule にメンバー要素を抽象定義することで、細かなノードの仕様やシナリオを実験者が実装することが可能である。抽象定義とは Dodule ではメンバの宣言のみ行い、利用者がインスタンスする際にメンバー要素を実装をすることである。下記の図 4.11 は Standard_NAT Dodule のメンバー要素である NAT の内側のノードを抽象定義し

た Abstract_NAT Dodule である。20 行目で X と抽象定義し、21 行目で X の第 1 インターフェースがネットワーク inn に参加している。そのため、Abstract_NAT Dodule を用いるためにはネットワークインターフェースを最低でも 1 つを持つノードクラスか Dodule が必要である。抽象定義の際に、**X**[] と配列化することで、利用時に複数のクラスを一度に指定することが可能となる。

```
1: module Standard_NAT(exn, X , X_num){
2:   network inn
3:   inn.ipaddrrange = "192.168.0.0/24"
4:
5:   nodeclass SNAT{//図4.9と同じ
6:         ....
7:   }
8:   snat = new SNAT
9:   attach snat[0].netif[0] exn
10:  attach snat[0].netif[1] inn
11:
12:  xnode[X_num] = new X
13:  attach xnode[0-X_num].netif[0] inn
14:
15:}
```

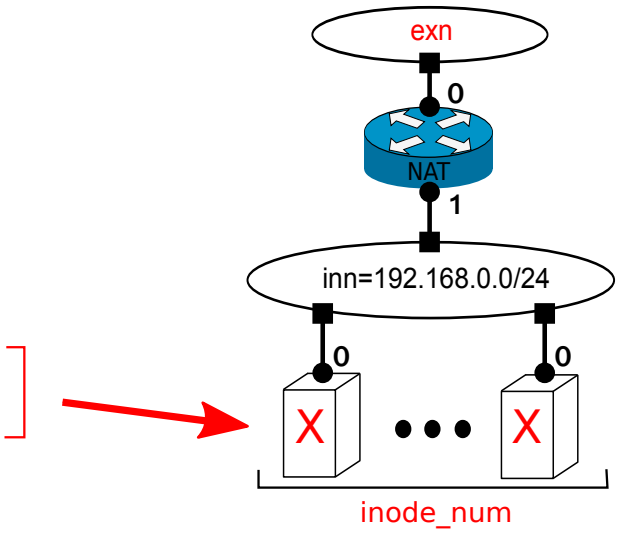

図 4.11: Abstract_NAT Dodule script

コンストラクタ

コンストラクタは引数の型や引数の数を変更することで 1 クラス内に複数宣言することができます。しかし、引数の型と引数の数が同じコンストラクタを 1 クラス内で複数宣言することはできないため、記述した場合はエラーとなる。複数または全てのメンバー変数の初期化や、特定のメンバー変数に対してはデフォルト値で初期化する際、コンストラクタを複数宣言することで可能となる。基本的なコンストラクタを用いた Dodule の記述方法は下記に示す。引数の数が異なる分だけ複数のコンストラクタを宣言することが可能となる。

```
Dodule Dodule名 {
    //引数n個
    Dodule名(引数,…){
        コンストラクタ本体
    }
    //引数m個
    Dodule名(引数,引数,…){
        コンストラクタ本体
    }
    ...
}
```

組み合わせ

既存のDoduleを利用して新しいDoduleを定義することが出来る。次の例では図4.9のStandard_NATを用いて多段NATを行うMultigate_NAT Doduleを作成している。

オーバライド

ノードとネットワーク及びそれらの属性と、Doduleのメンバー要素のアクセス方法は以下となっている。主に利用されるノードの属性としては、ディスクイメージの指定・ネットワークインターフェース・シナリオがあり、ネットワークの属性としてはIPアドレス範囲がある。

```
ノード変数.属性
ネットワーク変数.属性
Dodule名.メンバー変数
Dodule名.メンバー変数.属性
```

Doduleのメンバー変数と、それら属性を以下の方法をオーバライド可能とする。これにより、コンストラクタで扱っていない値を初期化することが可能となる。下記の例では、NATの内側にあるノードを変更する。Override_Nodeを用いてStandard_NATのメンバー要素であるinodeの属性をオーバライドする。オーバライドされたメインスクリプトの以降の行ではStandard_NATのメンバー要素であるinodeはOverride_Nodeと同じ属性を持つことになる。

```
1: import Standard_NAT
2:
3: Dodule Multigate_NAT(exn){
4:   network multigate_inn
5:   multigate_inn.ipaddrrange = "192.168.100.0/24"
6:
7:   node MNAT{
8:     netif[2] media gigabitethernet
9:     scenario{
10:      netiffit pathifscan
11:      callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
12:      callw "/sbin/ifconfig" self.netif[1].rname self.netif[1].ipaddr
13:      callw "/sbin/iptables" "-t" "nat" "-A" "POSTROUTING"
14:                  "-o" self.netif[0].rname "-j" "MASQUERADE"
15:      callw "/sbin/service" "iptables" "save"
16:    }
17:  }
18:  attach mnat.netif[0] exn
19:  attach mnat.netif[1] inn
20:
21:  nat_network = new Standard_NAT(multigate_inn, 3)
22:}
```

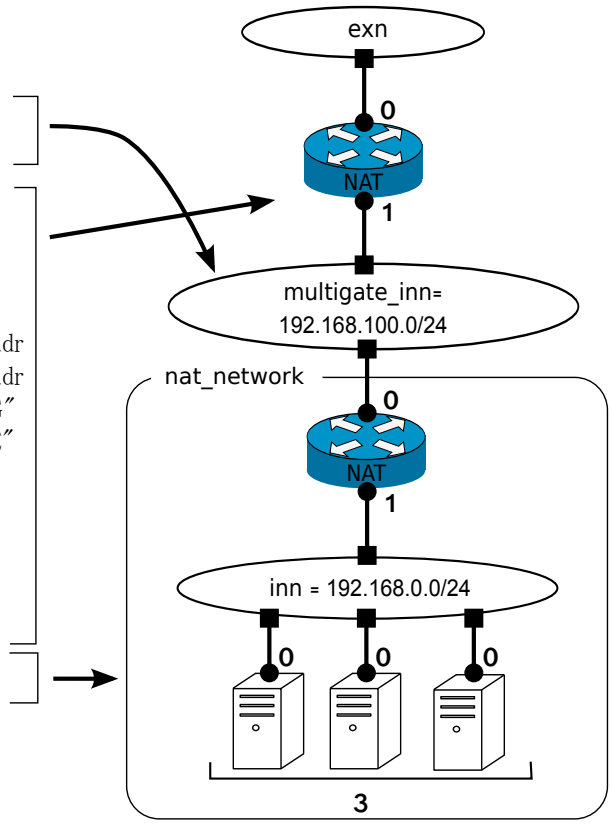

図 4.12: Multigate_NAT Dodule script

メインスクリプト

```
import Standard_NAT
nodeclass Override_Node{
    netif[2] media gigabitethernet
    scenario{
        netiffit pathifscan
        callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr
        print "node is override"
    }
}

Standard_NAT.inode =  Override_Node
```

4.5 システム構成

M 言語と SpringOS を用いた実験環境構築に関する方法として 2 つ考えられる。1 つ目は M 言語を評価する評価機を SpringOS の中に作成する。SpringOS へ入力する実験シナリオファイルを M 言語に変更する。これは、M 言語評価機に SpringOS の各モジュール群と通信を行う kuma と同様な機能を持つ必要がある。2 つ目は評価機を SpringOS の外に作成する方法である。K 言語のプリコンパイラとして M 言語から変換を行い、変換された出力結果を用いて実験を行う。この方法では SpringOS の中の機能を変更する必要がない。しかし、出力結果が K 言語であるため、現在の実験方法と変わらず実験を行えるが、K 言語が扱える能力を M 言語が扱うことは出来ない。本システムでは後者を採用し、構成を図 4.13 に示す。M 言語で記述されたシナリオファイルを変換機を MK translator(以下 MKt) に入力することで K 言語に変換される。

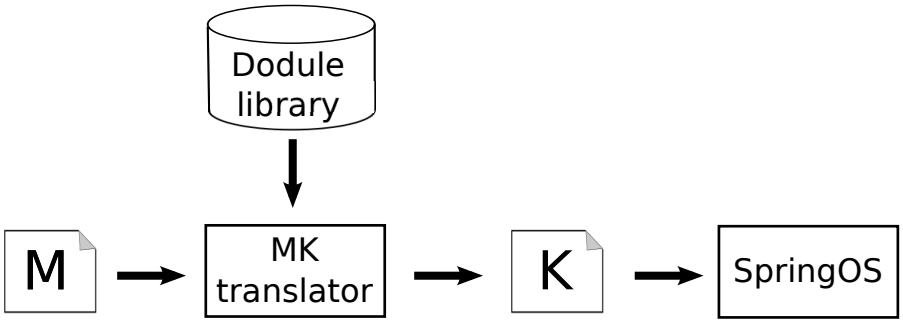

図 4.13: 本システムの構成

4.6 MK translatorの実装

MKtは実装するためのプログラミング言語としてruby 1.8.7を利用した。MKtはM言語で記述された実験シナリオファイルを1行づつを解析し、K言語の実験シナリオファイルへ変換する。処理の流れを図4.14に示す。MKtの言語処理系は大きく分けて、次の部分からなる。

- 字句解析：文字列を言語の要素（トークン、token）の列に分解する。
- 構文解析：token列がM言語の仕様に沿って記述されているかどうかのチェックを行う。
- 意味解析：記述された変数の型や文がM言語の記述仕様に沿っているかどうかをチェックする。
- コード生成：K言語に変換されたコードを生成する。

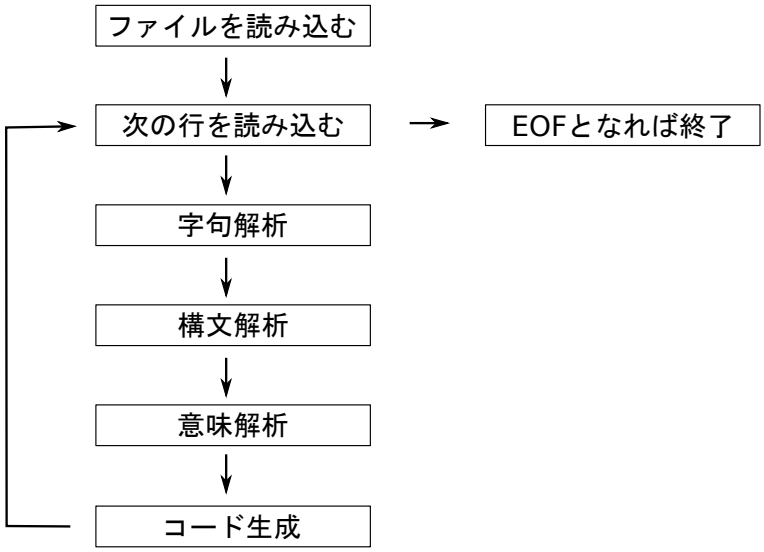

図 4.14: MKtの処理の流れ

4.7 評価

実験環境の作成に関して、既存研究であるSpringOSのK言語とM言語を比較する。比較する項目とその可否をまとめたものを表4.1に示す。

記述の再利用性は、K 言語では他の実験者が記述した実験シナリオファイルをテンプレートとして利用することが可能である。しかし、実験シナリオに取り組む際に、テンプレートファイルを書き換える必要があるため、柔軟にテンプレートを扱えるとは言えない。M 言語は Dodule を用いることで、実験環境を部品化して再利用可能である。また、Dodule は単純なテンプレートの展開とは違い、利用に応じたパラメータのインスタンス化が可能となっている。拡張性に関しては、実験者が既存の Dodule に機能を追加・変更した Dodule を新しく定義または再定義することが可能である。そのため、既存の Dodule 自体への変更を加えずに再定義することができる。

記述量に関しては、ある実験環境を記述する際、利用可能な Dodule がある場合実験記述量は少なくなる。しかし、利用可能な Dodule がない場合、K 言語と M 言語は記述量としてあまり変わりはない。

表 4.1: 実験環境の記述に関する比較

	K 言語	M 言語
再利用性	△	○
柔軟性	×	○
拡張性	×	○
ネットワーク実験の記述量	△	○

表 4.2: 実験環境の記述量に関する比較

	K 言語	M 言語
利用可能な定義モジュールあり	△	○
利用可能な定義モジュールなし	△	△

第5章　複数の実験環境の統合

大規模な実験を行う際、実験者一人で環境構築を行うことは考慮する事項が多いため、複数人のグループで実験を行うことが多い。大規模な実験環境を一括して構築することは問題が起こった時、問題点の発見が難しい。そのため、複数の実験者はそれぞれの実験環境を構築後に統合することが望ましい。

複数の大規模実証環境を利用した実験環境を構築するためには、それぞれの大規模実証環境に存在する制御機構の協調が必要であり、このために各大規模実証環境に存在するモジュール間の通信が必要である。実験環境内部の実現方法を隠蔽し、外部からは一般的なインターフェースでアクセスすることで、 複数の実験環境を利用した大規模な実験環境が構築できる。

5.1　実験環境統合の方向性

複数の実験環境を統合する際、方向性がある場合とない場合が考えられる。図 5.1 に実験環境統合の方向性の例を示す。図 5.1 の上図の方向性がある場合では、ある実験環境を構築しておき、他の実験環境からの参加を待ち受ける。クライアントサーバモデルと言っていい。クライアントがサーバに「接続要求」を送信し、サーバがそれに「接続応答」を返すことで統合が行われる。接続をするため、接続先の実験環境の細部を接続者は知る必要はない。

図 5.1 の下図の統合の際に方向性がない場合として、既にあるお互いの実験環境を組み合わせて新しい実験環境として構築する方法である。お互いの実験環境を組み合わせるため、お互いの実験環境の構成をしる必要がある。

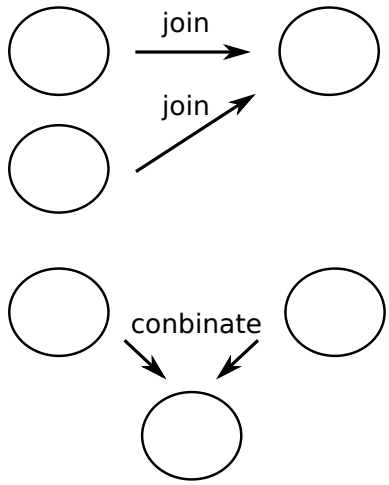

図 5.1: 実験環境の方向性

5.2 実験環境の抽象化

方向性のある統合の際、接続先の実験環境の細部は知る必要がない。そこで、接続先の実験環境を抽象化し、実験に関する説明と他の実験環境と統合箇所についての定義を行う。次の図 5.2 ではある実験環境を抽象化し、実験環境自体の説明と接続される箇所の説明を行ない、情報共有を行う。実験環境自体の情報共有としては、接続するとどのようなサービスが使えるか、データリンクのスペック、各レイヤの説明、リスタート方法などのオペレーション、電源容量、繋がらない時の連絡先、など外部に提供すべき情報として考えられる。また、接続される箇所を goat と定義する。この goat にはネットワークの接続として最低限必要なものとして、ネットワークの情報としてどの技術を用いて論理ネットワークを構築しているか、またそのリンク識別子を記述する。

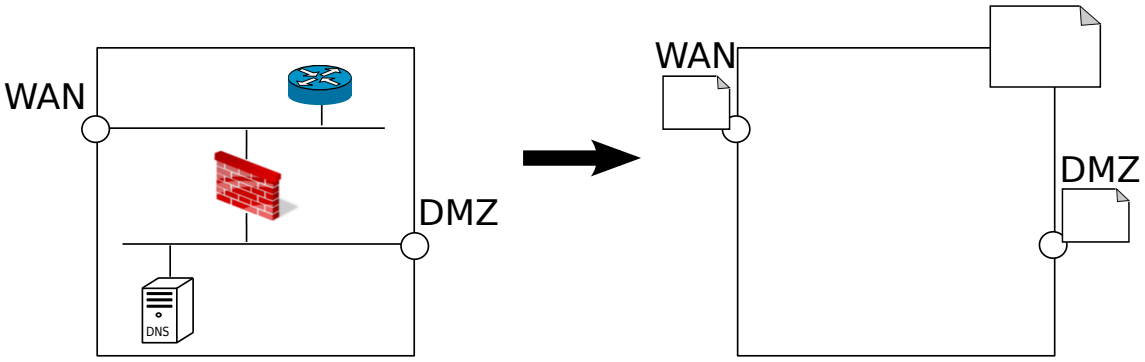

図 5.2: 実験環境の抽象化

5.3 実験環境統合例

統合の簡単な例として実験者が PPP over Ethernet(以下 PPPoE) プロトコルを用いて通信を行うやり取りを用いて説明する。

Point to Point Protocol（以下 PPP）は、2 点間を接続してデータ通信を行うための通信プロトコルである。この PPP の機能を Ethernet を通して利用するためのプロトコルを PPPoE という。PPPoE のやり取りを図 5.3 に示す。まず初めにでは、PPPoE client は PPPoE SERVER に ID,PASSWORD による認証を要求する。次にで、PPPoE SERVER は認証が行われれば、接続の許可と情報通知（IP）を送る。

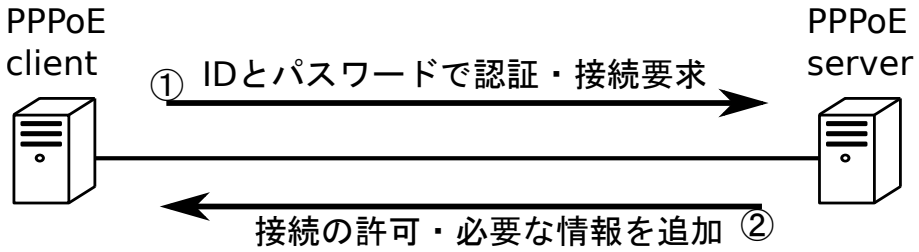

図 5.3: PPPoE の流れ

ある実験者が PPP プロトコルを用いて通信を行う。PPPoE で接続するために ID と PASSWORD が必要である。そのため、実験環境全体と接続箇所の goat を示す。

```
PPP{
    goat PPPoE{
        type ethernet
        ppp_id shota
        ppp_pass taro
    }
    DNS 192.168.0.1
}
```

図 5.4 では物理的な配線はすでに構築されていると仮定する。その場合、実験者はこの記述を参考に実験環境にアクセスする。id と pass は記述されているが、安全面を考えて記述を控えることも考えられる。

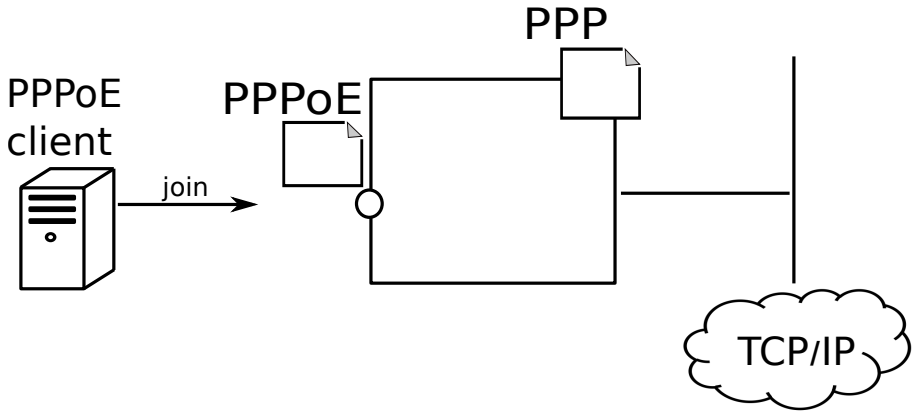

図 5.4: 接続例

第6章　おわりに

本研究では、容易な記述による実験環境の構築について述べた。既存の支援ソフトウェアによるネットワーク実験の構築では、ノードとネットワークの単位で設定を行ない、実験対象を含めた実験環境の詳細を全て記述する必要がある。そのため、上位レイヤの実験を行いたい実験者にとって記述は煩雑になる傾向がある。そこで、実験環境構築のための設定項目に注目し、実験環境の一部を定義モジュールという単位で扱い、実験者は必要に応じて定義モジュールを用いてネットワーク実験環境の構築を行う。提案手法を用いたネットワーク実験環境の評価と考察をした。その結果、記述の再利用性・簡略化・柔軟性の向上を確認し、容易な記述によるネットワーク実験環境の構築を可能とした。更に、実験環境同士の統合について議論も行い、基本的な例を基に説明を行った。

6.1　今後の課題と展望

6.1.1　動的な実験環境の変更

現在 SpringOS ではスクリプト言語を入力することで一括して実験の環境構築・遂行を行なっている。そのため、実験の一時停止やノードの追加などは行えない。そこで、SpringOS の kuma に実験遂行の流れを対話形式に行うインタラクティブモードを追加することで、実験者の任意のタイミングで実験環境を変更し、再実行が可能となる。

6.1.2　複数の実験環境統合の実装

本論文では、提案と考察のみだった複数の実験環境の統合の実装が課題として残っている。また、複数の実験環境を統合する話とは逆に、一つの実験環境をレイヤごとで実験者に割り当てることで、ネットワーク仮想化などのスライスによる作業分担が行える。

6.1.3　仮想ノードへの対応

仮想ノードを実験資源として追加することで、通常のノードと同様に扱うことができる。そこで、実験者のシナリオの要求に応じたリソース提供を行うことで、余分なリソースを余らせることなく物理マシンを利用することが出来る。

謝辞

本研究を行うにあたり、多くの方から多大なご助言やご助力をいただきました。それらの方々のご協力がなければ、本研究は成り立ちませんでした。心から厚くお礼申し上げます。

本研究を進めるにあたり、指導教員である篠田陽一教授には様々な助言、適切なご指導賜りました。心から深く感謝致します。また、助言を頂いた副指導教員である丹康雄教授、副テーマ指導教員である面和成准教授に感謝致します。

本研究室の知念賢一特任准教授と宇多仁助教には研究に関して活発な議論や多大なご指導を賜りました。心から感謝致します。

NICT の三輪信介博士，宮地利幸博士、太田悟史氏には研究に関して様々な助言、適切なご指導賜りました。心から感謝致します。

WIDE PROJECT の Deepspace One および NerdBox Freaks の櫨山寛章助教、宮本大輔助教、榎本真俊氏には本研究についての意見など、多くの助言をいただきました。ここに心から感謝致します。

篠田研究室の高野祐輝博士、安田慎吾氏、井上朋哉博士、LATT Khin Thida 氏、Nguyen Lan Tien 氏、Muhammad Imran Tariq 氏、明石邦夫氏、立花一樹氏、川瀬拓哉氏、鍛治祐希氏、大野夏希氏、田部英樹氏、向井雄一朗氏、岩橋紘司氏、加藤邦章氏、成田佳介氏、向井康貴氏、岩本裕真氏には研究以外でも普段の生活の面で支えていただきました。ここに心から感謝致します。

最後に研究や生活で支えてくれた家族へ心から感謝します。

参考文献

[1] StarBED Project: StarBED Project, http://www.starbed.org/

[2] PlanetLab: PlanetLab - An open platform for developing, deploying, and accesing planetary-scale services, http://www.planet-lab.org/

[3] Brian White, Shashi Guruprasad, Mac Newbold, Jay Lepreau Leigh Stoller, RobertRicci, Chad Barb, Mike Hibler, and Abhijeet Joglekar. Netbed: an integrated experimental environment. ACM SIGCOMM Computer Communications Review,Vol. 33, p. 27, July 2002.

[4] ns: The Network Simulator - ns-2, http://www.isi.edu/nsnam/ns.

[5] SpringOS : SpringOS, http://www.starbed.org/softwares/

[6] 宮地利幸. 大規模実証環境の実現と実験支援によるネットワークサービスの検証技術. PhD thesis, 北陸先端科学技術大学院大学, March 2007.

[7] 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 DSP-116, pp. 95-100, 2004.

[8] 知念賢一, 宮地利幸, 篠田陽一. Kuroyuri: ネットワーク実験記述言語処理系. コンピュータソフトウェア, Vol. 27, No. 4, pp. 4 43-4 57, 2010.

[9] Ken-ichi Chinen and Toshiyuki Miyachi and Yoichi Shinoda.A Rendezvous in Network Experiment—Case Study of Kuroyuri. In International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities(Tridentcom), March 2006.

[10] 宮地利幸, 知念賢一, 篠田陽一: “ SpringOS/VM: 大規模ネットワークテストベッドにおける仮想機械運用技術 ”, 情報処理学会研究報告書 2005-OS-99, ISSN 0919-6072, pp. 105-112, 沖縄,2005 年 5 月.

[11] 吉岡慎一郎，大規模ネットワーク実証実験設備における実験用資源の最適利用に関する研究， Master ’s thesis, 北陸先端科学技術大学院大学, 2011.

[12] 野中雄太. ネットワーク実験環境の保存と復元に関する研究. Master ’s thesis, 北陸先端科学技術大学院大学, 2008.

[13] 明石邦夫，大規模ネットワークの実証環境向け抽象化技術に関する研究， Master ’s thesis, 北陸先端科学技術大学院大学, 2010.

[14] 宮地利幸, 三輪信介, 篠田陽一， XBurner: XENebula を利用したトラフィックジェネレータプラットフォームの設計，インターネットと運用技術シンポジウム 2009 論文集, 情報処理学会シンポジウムシリーズ, Vol. 2009, No.15, pp.83-89, 金沢, 2009 年 12 月

[15] 宮地利幸, 三輪信介, 篠田陽一, XBurner を利用したトラフィック生成, マルチメディア，分散，協調とモバイル (DICOMO2010) シンポジウム論文集, 情報処理学会シンポジウムシリーズ, Vol.2010, No.1, pp. 304-313, ISSN 188200840, 下呂, 2010 年 7 月.

[16] 知念賢一, 三角真, 宮地利幸, 篠田陽一,StarBED におけるインタラクティブなノード制御，インターネットコンファレンス 2008 (IC2008), pp.5-14, 沖縄, ソフトウェア科学会, ISSN 1341-870X, 2008 年 10 月.

[17] 宮地 利幸, 三輪 信介, 知念 賢一， 一般的なプログラミング言語によるネットワーク実験駆動，DICOMO 2012，1520-1526，2012/07/05