マイクロコンピュータとワークステーション　68－4
（１９９１．　７．　５）

Network Software Environment とその利用事例

加藤 真紀子
日本サン・マイクロシステムズ株式会社

Network Software Environment (NSE) は、ソフトウェア開発を支援するための環境を提供するもので、比較的大規模なソフトウェア開発プロジェクトの各々の段階において生産性、品質などをを向上させることを目的としている。単に開発を支援するツールを提供するだけでなく、NSEの概念を導入し、新しいソフトウェア開発環境を構築する。このチュートリアルでは、まず、いままで抱えていた問題を簡単にレビューし、それら諸問題に対してNSEがどのような解を与えているのかを紹介する。さらに、簡単な具体例を、実際のコマンド操作の例と共に解説する。

Network Software Environment Tutorial

Makiko Kato
Nihon Sun Microsystms K.K.

Head Office
Kowa Nibancyo Bilg. 11-19, Nibancho, Chiyoda-ku, Tokyo 102, Japan

Network Software Environment (NSE) is a software development environment that can improve productivity and quality in all phases of software development projects. NSE provides not only development tools but the new concept to the software development project management. This tutorial review the problems to be solved in the current development style, then describe how NSE resolve these problems. And, show some actual command steps using the simple example.

1. はじめに

従来までのネットワーク・ソフトウェア開発環境では、開発に携わる人数が増え、プログラミング・プロジェクト自体の規模が大きくなるに従ってでてくる、種々の問題に悩まされてきた。ここでは、従来かかえていたいくつかの問題をあげ、Sun が提供している NSE (Network Software Environment) が、それらをどのように解決しようとしているかを簡単に紹介したい。

大規模なソフトウェア製品のプロジェクトが持つ特長としては、まず、膨大な数のファイルが存在する点、また、大規模になればなるほど、そのライフタイムが長く、開発後も長く管理が必要となる点、さらに、プロジェクト開発、リリース、マニュアル、保守、管理と、大プロジェクトにはもちろんそれに係わる人間も多くなり、効率のよい環境がプロジェクトのキーとなる点、などがあげらるであろう。

プロジェクトで管理するべきファイルは、ソース・ファイル、オブジェクト・ファイル、ライブラリ・ファイル、実行形式・ファイル、ドキュメント・ファイル等、種類も多様で、またその数も膨大である。そして、それらファイルを必要な情報と共に、秩序立てて管理し、手順よく最終目的の製品を構築することが、開発工程では必要となってくる。こういった作業をささえる道具として、UNIX で広く使われてきたのが make と sccs とよばれるツールである。

製品のリリースを繰り返していくと、当然マルチリリースをサポートしなくてはならず、それぞれのリリースバージョンの持つ複数のファイル管理が必要となるが、それには sccs が有効である。また、システムを再構築するために make を使えば、自動的に変更されたソースファイルを見つけだし、コンパイルし、システムを再リンクすることができる。これらの、sccs や make を有効に使えば、それなりにうまくプロジェクトを管理することは可能である。

しかし、これらのツールはいずれもファイル指向のツールであるため、完璧とはいえない。完全なソフトウェア開発サイクルを考えた場合、ソフトウェア製品が単にコードに関するファイル単位だけで構成されているわけではないことは明らかである。ソフトウェア製品の要素の中にはファイルの形にできるものばかりとは限らない。例えば、データ辞書などは、単純なファイルの中ではなくデータベースの中に埋め込まれるものである。

大規模なソフトウェア製品を構成するこうした要素の多様性は、ますます広がる傾向にある。

さらにもう一つ、ソフトウェア開発において、非常に大きな問題となってくるのが、複数の開発者によって平行開発していく上で起こる、スタッフ間の相互干渉の制御である。

同一ファイルに対して、同時に複数の開発者が変更を加る必要がでた場合に、sccs の排他制御を使って、同時変更は避けられるとしても、お互いに作業を進めなければならないため、結局それぞれ個人的なコピーをもって、変更を加え、後に矛盾が生じてまうことになりかねない。また、複数の開発者やモジュールで広く使われるようなファイルの内容が知らせもなく変更されていて、それが理由で、モジュール内に矛盾が生じ、開発がストップするなどといったこともある。逆に、個人的なコピー をもって干渉をさけて作業をしていると、別の開 発者の行なった機能拡張やバグ修正を反映させるのが困難であったりする。

共同開発では、基本的に開発者ごとに異なった部分を担当しているためある程度個々の独立した開発環境が必要である一方、ひとつのものを共同で作成しているという共同作業としての側面もあるため、理想的な開発環境を考えるのはなかなか困難なことである。

2. NSE の世界

これらの諸問題に対して Sun の提供する解決案が、NSE - Network Software Environment (現在はリリース1.3) である。従来はファイルとそれに対する処理として進められてきた開発環境・開発工程を、NSE では、開発環境を構成する要素を 'オブジェクト' と定義し、ファイル以外のも表現することができるため、管理・操作が、より柔軟なものになっている。ここでは、NSE が定義する開発環境を構成する 'オブジェクト'・'エンバイロメント' について説明する。

2.1 オブジェクト

NSE では、操作の対象となるものを、一般的に「オブジェクト」としてとらえており、それはファイルをはじめとして、開発途中に存在するすべての「操作の対象物」を総称する。そしてそのオブジェクトは、階層構造をもつことができる。すなわち、UNIX のファイルシステムのような tree を構築することができるわけである。

このオブジェクトの主要タイプとして、「ファイル」、「ターゲット」、「コンポーネント」の3つを定義している。

2.1.1 ファイル

NSE では、通常ファイル、ソースファイル、派生ファイルという3つの種類のファイルが区別して扱われる。

ソースファイルは、すなわちプログラムのソースコードのファイルであり最も重要なタイプである。このソースファイルについては、変更の履歴情報をすべてもつが、ディスク空間の節約のため、その差分のみで管理を行なっており、この点は sccs に共通している。NSE では vcs と呼ばれるバージョン・コントロール・システムが利用される。

通常ファイルについては、NSE のコンポーネント（後述）のメンバになれることを除けば、特別な機能は提供しない。

派生ファイルとは、コンパイラやリンク等のプログラムによって作成されるファイル、たとえばオブジェクトや実行ファイルなどのことで、これらはソースファイルから作成することができるので、バージョン管理を行なわない。

2.1.2 ターゲット

ターゲット・オブジェクトは、目的である生成物、それを構築するために必要なすべての派生ファイル、そしてその構築手順を記述した、「Makefile」によって定義され、いわばオブジェクトの集合を示すオブジェクトである。

目的とするライブラリなどは一つのターゲットであり、そのターゲットオブジェクトには、Makefile、それを構成するメンバのオブジェクト、ソースファイル、ヘッダファイルなど、構築に必要なすべてのファイルオブジェクトが含まれる。NSE は、このターゲットを常に最新で正しい状態に保持する。たとえば、常に最新の派生ファイル (.o など) が、ターゲットオブジェクトに含まれるように維持するし、またソースファイルの変更によって新たなヘッダファイルが必要となれば、そのヘッダファイルは自動的にターゲットに組み込まれる、といったぐあいである。

2.1.3 コンポーネント

コンポーネントとは、オブジェクトの、意味のあるひとかたまりをさす名称であり、NSE の世界では非常に重要な構成要素である。

たとえば、あるコンポーネントは主要なサブシステムを表し、その中にマイナーなサブシステムを表すコンポーネントを含む。さらに後者のコンポーネントには、プログラムを表す別のコンポーネントが含まれるといったぐあいに、再帰的な階層構造をとることができる。このようにコンポーネントはそのメンバとして他のコンポーネントを持つことができるため、ソフトウェア製品の構造を表すための自然なオブジェクトといえる。

コンポーネントのもっとも基本的なものは、ターゲットのみで構成されるものである。さらに複雑なものでは、ターゲットだけでなく、そのターゲットに関連するオブジェクト（例えばテキスト・データファイルや実行形式のテスト・ドライバ、ドキュメンテンション・ファイルなど）が含まれるものが考えられる。

つまり、変更・再構築・テストしたりするある特定の単位となるオブジェクトは、グループとして1つのコンポーネントとみなすことができる。どのオブジェクトを1かたまりのグループとして1コンポーネントにするかということは、プロジェクト・製品の構成状況に応じて決定することである。各コンポーネントはそれぞれ自由にメンバを共有することができる。例えば共通ヘッダファイルの実体は1つで、それを多くのコンポーネントが共有するということが可能である。

NSE は、ソースファイルのバージョンを管理するのと同様に、コンポーネントの各リビジョ

ンを維持するので、ファイル単位だけのリビジョン管理に比べて、より統一的な履歴管理をすることができる。この機能により、旧リビジョンのセットに含まれるファイルなどにいつでも簡単にアクセスすることができる。

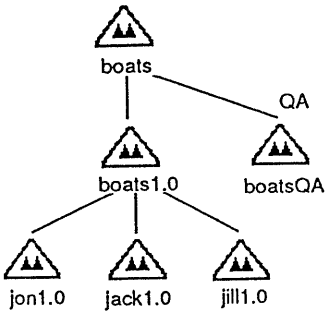
2.2 エンバイロメント

2.1 で述べたNSEのオブジェクト階層を利用すれば、複雑なソフトウェア製品の構造をよりよく管理することができる。しかし、オブジェクト階層だけでは、複雑な製品の開発に携わるプロジェクト・チーム・スタッフ間のやり取りで生じる問題に対処できるわけではない。

作業の対象となるものをオブジェクトと定義したが、それに対して、オブジェクトが存在するところ、それがエンバイロメント（環境）という概念であり、特定のオブジェクトに対する同時アクセスを管理するNSEの機能を実現している。エンバイロメントという概念が、現実にどのように利用されるのか、簡単な例をあげて説明しよう。

2.2.1 個人環境のエンバイロメント

エンバイロメントは、大抵の場合ターゲットを単位として作成される。



プロジェクト全体を示すエンバイロメントAは、製品Aを作成するために必要なすべてのオブジェクトが含まれ、これがおおもとのエンバイロメントとなる。この製品を構成するいくつかの機能、モジュールをそれぞれ担当するサブグループが存在すると、おおもとのエンバイロメントの中から担当するターゲットを指定し、各グループ用のエンバイロメントを作成すると、そのターゲットに必要なオブジェクトのみが存在するエンバイロメントが用意される。

例えば、ある製品の通信部分を担当するグループ、ユーザインタフェース部分を担当するグループで、それぞれ必要なライブラリ、ソースファイルの入ったエンバイロメントを持ち、ヘッダファイルなどは、グループに共通して利用される。このオブジェクトは、複数のエンバイロメントに存在することになる。ここで、もとになったエンバイロメントを親と呼び、そこから作られたエンバイロメントを子と呼ぶ。

従来の開発方法でいえば、各グループが必要となるディレクトリやファイルは、グループ独自のコピーとしてそれぞれソースツリーを作成する部分にあたるわけだが、そのやりかたと比較すると、いくつかの点ですぐれている。

エンバイロメントは、独自のコピーを持っているように見えるが、実際には、変更されないオブジェクトは共有されているのである。

また、独自のコピーを作る際には、どのファイル、どのディレクトリをもってくればよいのかがどうしても手作業になるわけだが、エンバイロメントは、あらかじめターゲットと、それを構成するオブジェクトとして定義されているオブジェクトをもってくればよいので、もれもなく、物理的なディレクトリ構成に制限されることもなく、管理が非常に楽である。

さらに、別々に開発を行なったあとに、独自のコピーを、どのように統合するかという点においてもNSEではすぐれた機能をもっているが、それについては後で述べることにする。

グループのエンバイロメントに加えて、そのグループメンバも個人別にエンバイロメントをもって、開発者どうしの同期、干渉問題を解決する。例えば、ユーザインタフェースのなかで、ツールａを担当する、ツールｂを担当する、それらに共有されるライブラリを担当するという具合にそれぞれの開発者がそれぞれのエンバイロメントを持って、その作業を行なうのである。

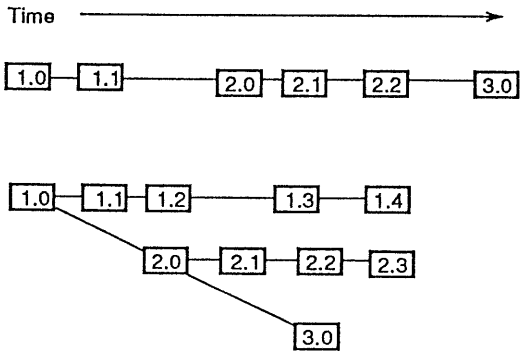
エンバイロメントは、個人個人で管理されるものと、グループ全体でアクセスされるものとあり、後者は、広くアクセス可能なようにパブリックエンバイロメントとして管理され、前者は、別の開発者によって干渉されないようにプライベートとして扱う、といったようにエンバイロメントごとの単位にアクセスコントロールができるようになっている。

それぞれのエンバイロメントで作業を行なうことをエンバイロメントをactivateするという表現を使う（コマンドは3章で詳しく説明する）。activateされたそれぞれのエンバイロメントをのぞいてみると、同じマシン、同じディレクトリ構造、ファイル名であっても、それぞれの変更が加えられた別々の内容をアクセスすることができる。例えば、

```
(a)% cd /usr/projectX/src/bin
(a)% ls
    commnd1/   comand2/   command3/
(b)% cd /usr/projectX/src/bin
(b)% ls
    comand2/   command3/   command4/
```

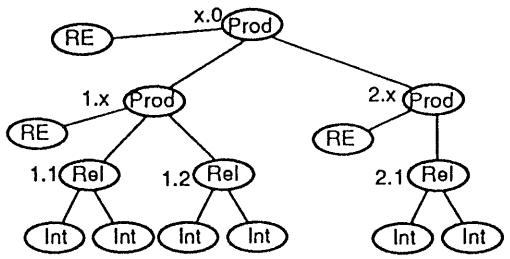
というように、別々の構成になっていたり、ソースファイルの内容も異なっていたりする。従って自分のエンバイロメントの中では他の開発者の変更を気にすることなく環境を保つことができ、プライベートなエンバイロメント内では、オブジェクトに自由に手を加えても、他の開発者のエンバイロメントに影響を及ぼさない。つまり開発者は、プライベートなエンバイロメントにおいて、好きなように変更を加え、ターゲットを最構築、テストすることができるわけである。従って変更を加えられた内容は、まず親のエンバイロメントに反映される。親に反映された内容は、すべての子エンバイロメントが必要に応じた時点でそのエンバイロメントを自分のエンバイロメントに反映させることができる。

2.2.2 開発工程ごとのエンバイロメント



次に、ひとつの製品のライフサイクルをながめてみると、開発、テスト、リリース、そしてバージョン1.0のテストがはじまるあたりから、バージョン1.1の開発サイクルがはじまり、さらに、ある時点でバージョン1.0の製品が出荷されその後、サポート用にそのソースを管理していかなければならない、というように、いくつかの部署が、同じ製品を構成するファイル群に、異なる目的で同時にかかわるということがおこってくる。

このような場面でも、エンバイロメントは有効に利用される。図のように、各部が、それぞれ、すべてのオブジェクトを含むエンバイロメントを保有することで、無駄なく資源を管理していくことができるわけである。



環境階層の最上位に位置するエンバイロメントは、リリース環境でここでは完成品のテストが行なわれ、メーカーや顧客にその製品がリリースされる。リリース環境には製品のオブジェクト階層の最上位のコンポーネントが含まれ、これらのコンポーネントには他のすべてのコンポーネントがサブコンポーネントとして含まれている。新しい大きなリリースの開発を開始する場合、リリース・エンジニアは現在のリリース環境の下に新しいリリース環境を作成し、古い環境から新しい環境のコンポーネント・リビジョンをコピーすれば、新しいリリースの開発ベースが得られることになる。

2.3 バリアントと実行セット *(variants & execset)*

エンバイロメントは、異なったハードウェアのアーキテクチャーをサポートしなければならない環境や、異なったオペレーティングシステムのバージョンに対応する製品を管理するために、バリアントと実行セットという機能を持っている。

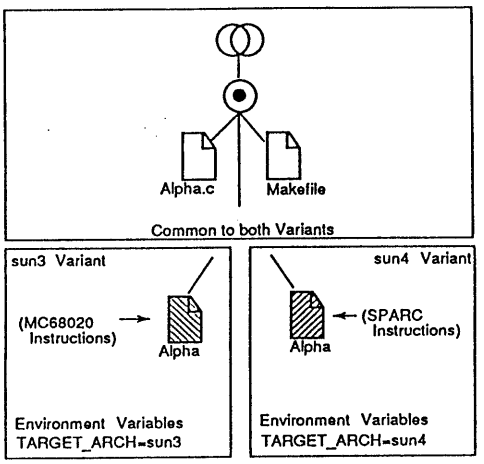
ターゲットとするアーキテクチャーやオペレー

ティングシステムのバージョンが異なっても、ソースファイルは共有できる。しかし、それをmakeしたときに派生する.oファイルや実行ファイルは異なるため、多くのアーキテクチャーを1つの環境でサポートするのは困難である。

これを解決するために、エンバイロメントは、バージョンやアーキテクチャーを属性としてもつことができ、これをバリアントと呼んでいる。いわば、activateされたエンバイロメントに特有のUNIXの環境変数をあらかじめ定義しておくことができるというような意味である。

同じAというエンバイロメントでも、mc68020というバリアントとSPARCというバリアントは、別々にactivateすることができ、そこでmakeした派生ファイルは、もちろん異なるわけである。この機能によって、異なるアーキテクチャーなどのサポートが大変容易になる。

またデバッグ用のバイナリや、パフォーマンス測定用のバイナリをそれぞれ作成するためのバリアントというような利用もできる。



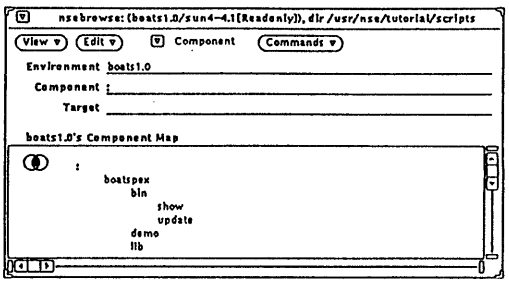
さらに、あるバリアントで、システムを構築するときに利用されるコンパイラやコマンド、ライブラリが、別のバリアントでは、異なったセットであるようなこともありえる。

例えば、mc68020のバリアントでは、普通のC compilerを利用するが、SPARCのバリアントでは、cross compilerを利用する、といったぐあいである。このようなことをサポートするのがexecsetである。あらかじめシステムbuildに利用されるべきコンパイラなどの必要なバイナリ、ファイルのセットを定義しておくことで、ソースは、まったく共通に利用され、それぞれ指定された道具を使って、システムbuildが行なわれる。

プロジェクトシステムの管理者にとっては、そういった環境設定が簡単に、かつシステマティックに行なえることが大変有効であり、開発者にとっては、正しいエンバイロメントをactivateするということだけで、すべての環境設定ができてしまうのでとても有用であるといえる。

3 NSEの使用例

NSEは従来のUNIXスタイルのコマンド・インターフェースからでも、NSEのbrowserと呼ばれるグラフィカル・インターフェースからでも利用できる。図に示すように、NSEのbrowserはOPEN LOOKのグラフィカル・ユーザー・インターフェースのウィンドを実現しており、このウィンドウを使うためになにか特別に学ぶ必要はない。



ここでは、具体的にboatspexというアプリケーションの開発を何人かの開発者によって進めるというという例をあげて解説する。ただし、ここでは全体的な流れを紹介するだけにとどめ、細かいコマンドや手順などは省く。また、作業はウィンドウベースでもすすめられるが、ここではコマンドラインのインタフェースで説明する。

3.1 開発グループの骨組みを作る

新しい製品の開発をスタートする時まず1番初めにすることは、元となる新しい環境を構築し、その中でコンポーネントを定義することである。この作業はシステムの管理者によって行なわれる。ここでは、まったく何も作られていない状態からはじめる例をあげているが、すで

にソースツリーが存在するようなプロジェクトをNSEに移行するためのツールは別に用意されている。まず、おおもととなるエンバイロメントboatsを作成する。

```
% nseenv create boats
```

干渉を避けるためboatsエンバイロメントをロックしてから環境をactivateし、その中でコンポーネントやターゲットを作っていく。activateすると、すべての環境がととのった別のshellが一つたちあがり、(boats)%というプロンプトで、そのエンバイロメントの中にいることを示している。

```
% nselock -e boats on
% activate boats
(boats)% nsecomp add :Comp boatspex
(boats)% nsecomp add :boatspex Comp lib\
        bin demo
```

この時点で、リビジョンを付け、おおもとのエンバイロメントとして用意できたことになる。

```
(boats)% preserve -c "Init BoatSpex" :
```

次に今回のバージョンboats1.0用のエンバイロメントをboatsの子エンバイロンメントとして構築し(aquire)、それが開発者の親になるよう、ここでリビジョンを付ける。

```
% acquire -C -c boats1.0 -p boats :
% activate boats1.0
(boats1.0)% preserve -c "Init BoatSpex1.0" :
```

3.2 個々の開発者の作業

boats1.0エンバイロメントを親として、それぞれの開発者用のエンバイロメントを構築し、作業はその中で進められれる。ここでは、:boatspex:binコンポーネントのみをもつ子エンバイロメントを作成し、それをactivateして、作業を進めている。ファイルの変更、作成などは、すべてvcsを使い、バージョンコントロールの対象とし、実際のファイルを変更すると同時に、コンポーネントの構成の変更も行なっている(nsecomp)。変更を終了すると、reconcileコマンドで、親にその変更を統合する(reconcile)。

```
% acquire -c jack1.0 -p boats1.0 :boatspex:bin
% activate jack1.0
(jack1.0)% show.c を作成、テスト
(jack1.0)% vcs checkin -c "Init Ver show.c"
(jack1.0)% nsecomp add :boatspex:bin Comp\
        show
(jack1.0)% nsecomp add :boatspex:bin:show
(jack1.0)% reconcile -c "Trivial show"\
        :boatspex:bin
```

3.3 変更を親環境に統合

それぞれの担当部分の別々のエンバイロメントで作業をするめるが、nsenotifyを使って、別の開発者によって変更があったことをすぐに知ることができるように設定することができる。下の例ではjackはjillの開発しているライブラリを使っているので、機能拡張を即座に自分の環境に反映させる必要がある。そのため、:boatspex:libというコンポーネントに変更があったらmailで知らせがくるように設定しておいて自分の作業をすすめている。開発者jackは、updateというコンポーネントを追加してreconcileする。

```
(jack1.0)% nsenotify -e boats1.0 register\
        reconcile-to Comp :boatspex:lib
(jack1.0)% 諸作業
(jack1.0)% reconcil :boatspex:bin:update
```

開発者jillは、ライブラリlibboat.aを変更し、親にもどす。

```
(jill1.0)% acquire :boatspex:lib
(jill1.0)% vcs checkin lib.mk
(jill1.0)% reconcile :boatspex:lib
```

すると、ここでjackにmailが届けられる。知らせを受け取ったjackは、jillの変更を自分の環境に反映させるため、resyncを行なう。

```
(jack1.0)% resync :boatspex:bin:update
```

3.4 衝突の解決

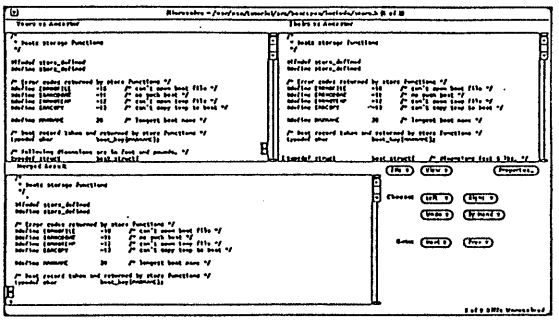
ライブラリlibboat.aの同じソースファイルをjillとjackがそれぞれの環境で並行で変更を行なったような場合、2人がreconcileをしようとすると、後でreconcileを試みた方は、衝突がおこったことが知らされる。

```
(jill1.0)% store.c store.h を変更し
        libboat.a を更新。
(jill1.0)% reconcile :boatspex:lib
```

(jack1.0)% store.c store.h を変更し
libboat.a を更新。
(jack1.0)% reconcile :boatspex:lib

jack は :boatspex:lib を reconcil しようとするが、jill の行なった変更と衝突してしまうため、その旨のメッセージがでて、reconcile できない。2人の変更を併合した後に reconcile をやりなおす必要がある。

(jack1.0)% reconcile :boatspex:lib
(jack1.0)% fileresolve &
(jack1.0)% reconcile :boatspex:lib



fileresolve は、その前に行なった reconcile の結果情報を自動的に調べ、必要なソースファイルに関して、図のようなマージツールを立ち上げてくる。

fileresolve ツールの左上のウィンドウには、自分の編集したファイルが追加、削除、変更のマーク付きで表示される。右上のウィンドウには、先に他の開発者が変更したファイルが同じようにマーク付きで表示される。そして、衝突のない部分を自動的にマージし、開発者の判断の必要な部分だけがマークされた内容が左下のウィンドウに表示される。上の2つのウィンドウで、どの部分が共通で、どの部分の変更に実際の衝突が生じているかが一目でわかるので、「右を選択」、「左を選択」、手で入力、といった、ほとんど機械的な作業でマージ（併合）作業を進めることができる。このマージ作業をつぎつぎに済ませ、ターゲットを再構築し、テストをすれば reconcile をパスすることができる。

以上に述べたような手順で、進められる。

5. あとがき

NSE は、いくつかの技術によって支えられている。まず、NFS (分散ファイルシステムによって、ネットワークを介するファイルの共有を可能にしている。さらに、NSE のエンバイロメント、コンポーネント、ユーザ、システムなどの情報をネットワーク上のさまざまなシステムから共通にアクセスできるようにするためにNIS (YP) システムや RPC (リモート・プロシージャ・コール)を利用している。複数のエンバイロメントで、同じファイルシステムをアクセスしつつも、内容の違った独自の環境をユーザに提供するために、TFS (Translucent File System)を利用しており、これは透過的にファイルシステムをマウントすることを可能にするものである。

これらの技術によって、NSE はソフトウェア開発に新しい概念を持ち込んでおり、大規模なプロジェクトを進めていくうえで、非常に頼もしいシステムといえるであろう。

謝辞

全般的にご指導いただいた、研究開発部・大川恵子氏には、多大な時間を割いていていただき、深く感謝いたします。

参考文献

NSE 1.3 Command reference
NSE 1.3 Administration Guide
NSE 1.3 User's Guide
NSE 1.3 Overview and Tutorial
NSE 1.3 OpenWindow Browser