プログラミング言語 18－1
（１９８８．１０．１４）

Ｃ言語によるソフトウェア開発の管理システム ＣＯＲＯＮＡ の設計

吉沢　克典　（山梨大学工学部）　　徳田　雄洋　（東京工業大学工学部）

本研究で実現を目指している開発管理システムＣＯＲＯＮＡは、Ｃ言語を用いてソフトウェアの開発を行う際に、その開発およびソースコードを管理するシステムであり、開発支援ツールの１つである。本論文では、Ｃ言語の特徴や、それを開発に用いる際の問題点、また複数の人間で開発を行う際の問題について述べ、その対策および解決法とともにＣＯＲＯＮＡの仕様について考える。また、ＣＯＲＯＮＡを実現する際に考えなければならない、いくつかの課題についても述べる。なお、ＣＯＲＯＮＡはＶＡＸ－１１／７８５[*1]のＵＮＩＸ[*2]４．３ＢＳＤ上に実現する。

On the Design of a Software Development Management System CORONA for Softwares Developments in the C Language

Katsunori YOSHIZAWA (Dept. of Computer Science, Yamanashi University
4-3-11 Takeda, Kofu-shi, Yamanashi-ken, 400, Japan)
and
Takehiro TOKUDA (Dept. of Computer Science, Tokyo Institute of Technology)

We propose a software development management system CORONA for writing and modifying software in the C language. The facilities of the CORONA system are source code database, source code inquiry, perturbation analysis, concurrency control, and protection mechanism. These facilities are designed especially for the development of software in the C language. In this paper we focus on the design and implementation issues for implementing the CORONA system on VAX-11/785[*1] under 4.3BSD UNIX[*2] operating system.

[*1] ＶＡＸ－１１はＤＥＣ社の登録商標である。
[*2] ＵＮＩＸはＡＴ＆Ｔベル研究所の登録商標である。

1．はじめに

近年ＵＮＩＸの普及にともなって、Ｃ言語が開発に使用される機会が年々増えてきている。いまや、ＵＮＩＸ上でＣ言語を用いて大規模な開発を行うことは、決して珍しくない光景である。

このような状況下で、ＵＮＩＸ上にもさまざまな開発支援ツールが用意されてきている。その中でも特に代表的なものは、Ｓｙｓｔｅｍ Ｖ上のＳＣＣＳ[4]や、４．３ＢＳＤ上のＲＣＳ[8]といったソースコード管理システムであろう。これらのシステムは、ソースコードのバージョン管理を行うものであるが、テキストファイルを管理の対象にしているため、プログラムばかりでなく、ドキュメントなども扱うことができ、汎用的であるといえる。しかしその反面、ある特定の種類のものに対して、専門的な管理を行うことはできない。これに対して本論文で提案するＣＯＲＯＮＡは、管理の対象をＣ言語に限定して、Ｃ言語に対する専門的な開発管理を行おうとするものである。

2．Ｃ言語のモジュール性

Ｃ言語の特徴の１つとして、モジュール性があげられる。つまり、複数のソースファイルが１つのソフトウェアを構成するわけである。これは、複数の人間で開発を行うのに適している。それぞれの開発者が分担を決めてソースファイルを作成し、最終的にそれらを結合して１つのソフトウェアを作り上げる。

しかし、Ｃ言語は各ファイル間の依存性が強い。例えば、あるファイル内で使用している変数が、他のファイルで定義されていたり、他のファイルにある関数を使用したりする。特にヘッダファイルなどは、他のファイルへの影響が非常に大きい（図１）。これは、決して特別な例ではなく、Ｃ言語で大きなソフトウェアを作る場合、ほとんどこのような構造になる。また、このようにした方が記述が簡潔にでき、実行効率もよい。

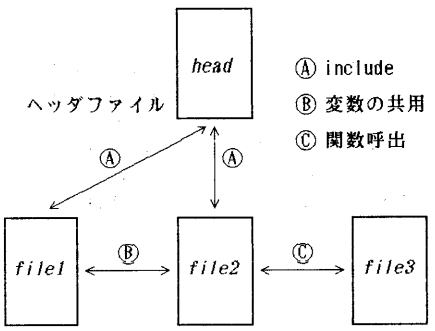

図１　ファイル間の依存関係

ところが依存関係が強いために、あるファイルを修正すると、その影響が修正したファイル内にとどまらず、他のファイルにまで及ぶことが少なくない。従って、Ｃ言語で開発を行うと、このような問題が開発の能率を落とす原因にもなり得る。あるファイルを修正したのに、他のファイルでその修正に関連する部分を見落としていたり、修正の影響を考えていなかったりする。つまり、ファイル間に矛盾が起こってしまうのである。

しかも、複数の人間で開発を行う場合は、さらに問題が深刻になる。ある開発者が、担当のファイルを変更したことを他の開発者に知らせなかったとすれば、その影響が及ぶ側のファイルの担当者は頭を痛めることになるだろう。ファイルをまったく修正していないのに、いままで正常であった部分で問題が起こるなどというようなことは、いくらでもあり得ることである。

ＣＯＲＯＮＡの基本的な目標は、このような影響を調べ、開発者に知らせることによって、開発の効率をあげることである。

3．グループ開発の一般的な問題点

一般的に大きなソフトウェアは、複数の開発者によるグループ開発によって作成される。複数の人間がファイルを扱うわけであるから、個人で開発する場合には起こらないようなさまざまな問題が現れてくる。

3．1　ファイルの保護

複数の人間がファイルを扱うため、グループ開発では、特にファイル保護の問題が重要である。うっかり大切なファイルを消してしまったり、自分の担当でないファイルを勝手に修正したりするなどの問題が起こる。このような問題は、Ｃ言語での開発に限らず、グループで開発を行う場合には、どうしても避けられない問題であろう。

ＣＯＲＯＮＡは大きなソフトウェアの開発に対応するために、特別な環境などを用いてファイルの保護を確実に行い、このような問題をサポートする機能を持つ。

3．2　開発者間のコミュニケーション

グループ開発を行う上で非常に重要なのは、いかに開発者の間でコミュニケーションをとるかということである。最初にも述べたように、自分の担当のファイルさえ作成すればよいというわけではない。例えば自分の行った修正の影響が、他のファイルにどのように及ぶかなどといったことを、他の開発者にも知らせる必要がある。また、他の開発者がいま何を行っているかなどということも、重要な情報のひとつである。

ＣＯＲＯＮＡは開発者間のコミュニケーションを

助ける機能も持つ。

4．CORONAの設計

大きなソフトウェアを、C言語を用いて複数の人間で開発する場合には、さまざまな問題が起こり得ることを述べてきた。CORONAはこれらの問題に対応し、開発を管理するシステムである。ここでは、その仕様について考える。

4．1　CORONAの役割

単純に言えば、CORONAは図書館などの窓口である。まず最初に、ソフトウェアを構成するファイルのソースコードを登録する。CORONAはそれらのソースコードをデータベースとして保存する。このデータベースに登録されているソースコードが公式版であると考えればよい。

その後、開発者があるソースコードを修正したいときには、CORONAを通してデータベースから目的のソースコードを取り出すわけである。開発者はこうして生成されたソースコードを修正したりテストしながら、ソースコードを新しいものにする。ソースコードの修正が一段落したら、それを新たな公式版として保存する。それには、修正されたソースコードを、CORONAを通して再度データベースに登録する。

なお、開発者が行うソースコードの修正やテストなどの作業に対しては、CORONAはまったく関知しない。ただし、これらの作業を行っているときに、CORONAを通してデータベースから情報を取り出すことはできる。

このように、開発者はCORONAを通してデータベースにアクセスし、ソースコードを生成したり登録することによって、ソフトウェアを開発していくことになる（図２）。

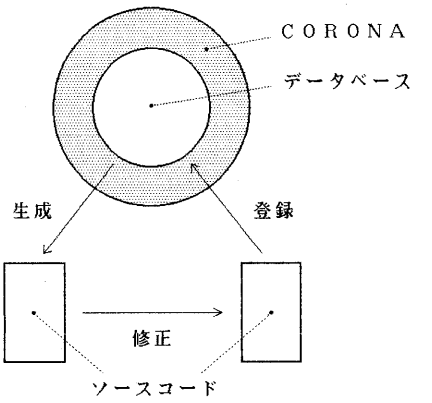

図２　CORONAの役割

4．2　データベースの構造

CORONAでは１つのソフトウェアを構成するソースコードはすべて、１つのデータベースとして扱う。

データベースは階層的な構造を持つが、まず目的となるソフトウェアをプロジェクトと考え、さらに、そのソフトウェアを構成するそれぞれのソースコードを、モジュールと考える。つまり、１つのプロジェクトに対して１つのデータベースが存在し、その中には、そのプロジェクトを構成する複数のモジュールがあるという構造である。なお、データベースに対するソースコードの生成や登録は、このモジュール単位で行う。

このような階層的な構造は、UNIXが持つディレクトリ構造をそのまま用いて、開発者が作業を行う作業ディレクトリの下に実現する（図３）。

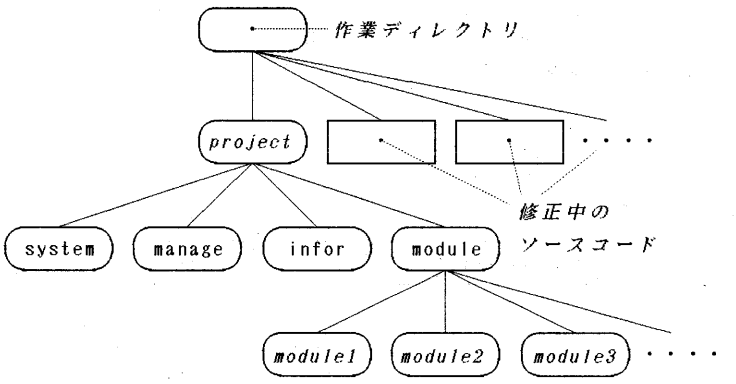

図３　データベースの構造

４．３　サブモジュール単位への分割

ＣＯＲＯＮＡの基本機能は、修正の影響を調べることであるが、あるモジュールの修正による影響を効率よくとらえるには、１つのモジュールをさらに細かく分割し、管理するとよい。Ｃ言語の１つのソースコードの内部は、機能によって、さらにいくつかの部分に分けることができる。その機能ごとに、以下のように分割されたそれぞれの部分を、サブモジュールと呼ぶことにする。

・ｉｎｃｌｕｄｅサブモジュール：
　　ｉｎｃｌｕｄｅファイル記述部分
・ｄｅｆｉｎｅサブモジュール　：
　　ｄｅｆｉｎｅ宣言部分
・型定義サブモジュール　　：
　　型定義部分
・データ定義サブモジュール　：
　　変数の定義部分
・データ宣言サブモジュール　：
　　輸入変数の宣言部分
・局所データ定義サブモジュール：
　　他のモジュールに独立なデータの定義部分
・関数定義サブモジュール　　：
　　関数定義部分
　　（各関数ごとにファイルに分ける）
・関数宣言サブモジュール　　：
　　輸入関数の宣言部分

なおここで、「輸入」という言葉は、他のモジュールで定義されているものを、使用するという意味である。つまり、輸入変数とは、そのモジュールで使用すると宣言しているが、別のモジュールで定義されている変数のことである。またその逆に、そのモジュールの中で定義しているが、別のモジュールでも使用される可能性のある変数のことを輸出変数という[2]。

これらのサブモジュールは、それぞれのモジュールのディレクトリの下に実現する（図４）。なお、モジュールのディレクトリの下に作られるのは、サブモジュール用のディレクトリであって、サブモジュールの本体などは、さらにそのディレクトリの下におく。

また、関数定義サブモジュールだけは、さらに各関数ごとに分割して管理する。そこで、関数定義サブモジュールのディレクトリであるｆ＿ｄｅｆディレクトリの下に、各関数用のディレクトリを作り、関数の本体などは、それぞれの関数用のディレクトリの下におく。

サブモジュールへの分割は、モジュールをデータベースに登録する際に行う。また同時に、それぞれのサブモジュールから必要な情報を取り出し、情報テーブルとしてファイルの形で保存しておく。こうしてサブモジュールに分割して管理しておけば、情報テーブル内の情報を検索することによって、修正の影響を効率よく調べることができる。

なお、ＣＯＲＯＮＡが調べる影響のうちで主なものは以下の通りである。

・ｉｎｃｌｕｄｅの影響
　あるモジュールを修正した場合は、他のモジュールのｉｎｃｌｕｄｅサブモジュールを調べ、その修正されたモジュールをｉｎｃｌｕｄｅしていたら、影響が及ぶと判断する。
・変数の修正の影響
　あるモジュールのデータ定義サブモジュールを修正した場合、自分自身あるいは他のモジュールのデータ宣言サブモジュールを調べ、修正された変数を輸入しているときには、変数の型などを調べる。逆に、データ宣言サブモジュールを修正した場合は、自分自身あるいは他のモジュールのデータ定義サブモジュールを調べ、定義されている変数と矛盾がないかを調べる。
・関数の修正の影響
　あるモジュールの関数定義サブモジュールを修正した場合、自分自身あるいは他のモジュールの関数宣言サブモジュールを調べ、修正された関数を輸入しているときは、影響が及ぶと判断する。また同時に関数の型や引数などを調べる。逆に、関数宣言サブモジュールを修正した場合は、自分自身あるいは他のモジュールの関数宣言サブモジュールを調べ、定義されている関数と矛盾がないかを調べる。

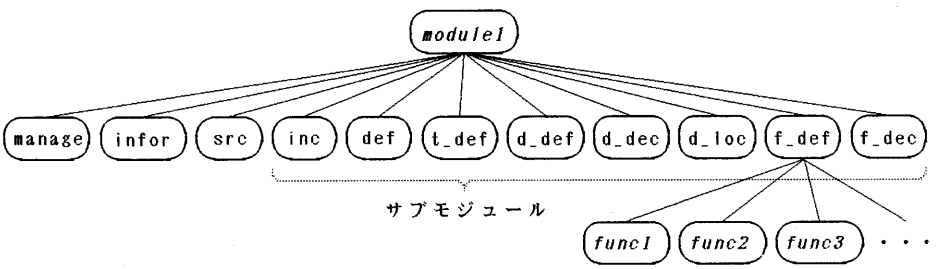

図４　サブモジュールのディレクトリ構造

４．４　ファイルの保護

グループ開発を行う場合は、ファイルや情報などの保護が重要な問題であることはすでに述べたが、ＣＯＲＯＮＡでは、特別な環境を設定してファイルの保護を行う。

先にも述べたように、データベースへのアクセスは、かならずＣＯＲＯＮＡを通して行うのであって、エディタなどで、直接データベース内のファイルを修正することは禁止しなければならない。これを許した場合、データベース内の情報に矛盾が生じてしまう。この問題は以下の方法を用いて解決する。

まず、プロジェクト全体の責任者を１名決定する。この責任者をプロジェクトマネージャと呼び、このソフトウェアの開発に関してすべての権利を持つ。ＵＮＩＸでのスーパーユーザのようなものであるが、ＣＯＲＯＮＡは、最初にこのプロジェクトのデータベースを作成したユーザを、プロジェクトマネージャであると決定する。

そして、データベース内のすべてのディレクトリやファイルの所有者をプロジェクトマネージャにして、さらにファイルには保護モード４００を設定する。これによって、データベース内のファイルを直接修正することはできなくなる。

しかしこれだけでは、ファイルを消去することはできる。そこでそれぞれのディレクトリに保護モード７００を設定することによって、ファイルを消去できなくする（図５）。

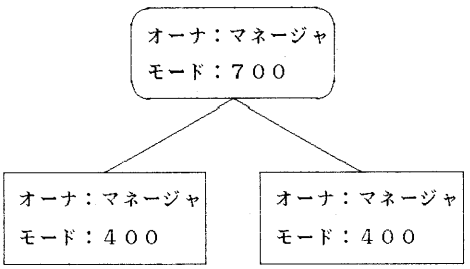

図５　ディレクトリとファイル

ただしこのままでは、プロジェクトマネージャ以外の開発者が新しくなったソースコードを登録しようとしても、ＣＯＲＯＮＡはデータベース内のファイルを書き換えることができない。これは、ディレクトリの保護モードが７００になっているため、作業ファイルを作ったり、元のファイルを消すことができないためである。これでは、プロジェクトマネージャ以外の開発者はＣＯＲＯＮＡを通しても、ファイルを更新できないことになってしまう。

そこで、特別な保護モード４０００を使用する。このモードが設定されているファイルは、誰が実行しても、実行中だけは、そのファイルの所有者が実行していると見なされるというものである[7]。従って、ＣＯＲＯＮＡの所有者をプロジェクトマネージャにして、このモードを設定すれば、問題は解決できる。ＣＯＲＯＮＡを通してのみ、データベースにアクセスできるのである。

なお実際は、ＣＯＲＯＮＡに直接このモードを設定するのではない。システム上にいくつものプロジェクトが存在し、何人ものプロジェクトマネージャがいた場合、それぞれのプロジェクトマネージャに対してＣＯＲＯＮＡを与えていたのでは、ディスク領域のむだ使いもはなはだしい。そこで、プロジェクトマネージャにはＣＯＲＯＮＡの代わりに、非常に小さなファイルを与え、このファイルに先ほどの保護モードを設定する。このファイルは、単に実際のＣＯＲＯＮＡを呼び出すだけのファイルであり、これをシステムインターフェイスと呼ぶ。

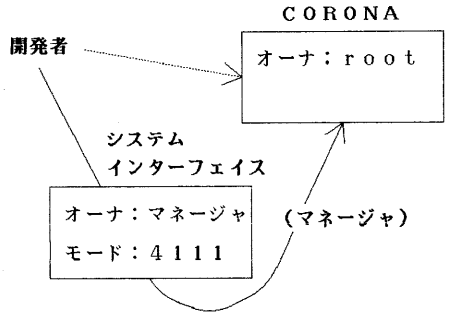

図６　システムインターフェイス

従って、開発者は直接ＣＯＲＯＮＡを呼び出すのではなく、このシステムインターフェイスを通してＣＯＲＯＮＡを起動することになる（図６）。また、システムインターフェイスも、やはりＣＯＲＯＮＡという名前であるので、サーチパスを利用すれば、システムインターフェイスを意識しなくてすむ。なお、システムインターフェイスはデータベース内のｓｙｓｔｅｍディレクトリにおかれる。

４．５　アクセス権の設定

ＣＯＲＯＮＡにはデータベースにアクセスするためのサブコマンドをいくつか用意するが、これらのサブコマンドはすべての開発者が自由に実行できるわけではなく、実行の対象となるモジュールなどに対して、アクセスする権利を持っている必要がある。

先ほど述べたような特別な環境や保護モードをデータベースに設定しておくことによって、ファイルに対するアクセスの権利は、ＵＮＩＸ上の保護モードにまったく関係なくＣＯＲＯＮＡで独自に設定し、管理することが可能になる。

なお、アクセスの権利には、以下に示す３種類のものがある。

・ｍａｎａｇｅ権　：　管理を行うことができる権利
・ｗｒｉｔｅ権　：　修正を行うことができる権利
・ｒｅａｄ権　：　参照を行うことができる権利

ＣＯＲＯＮＡでは、プロジェクトの責任者を１名設定することはすでに述べたが、さらに各モジュールに対して、それぞれ１名ずつの責任者を設定する。これらの責任者をモジュールマネージャと呼ぶが、プロジェクトマネージャがプロジェクト全体に対して特権を持っているのに対して、モジュールマネージャは、それぞれの対応するモジュールに対してのみ、特権を持つことができる。このようなプロジェクトマネージャや、モジュールマネージャが有する権利がｍａｎａｇｅ権であり、プロジェクトやモジュールの管理に関わるサブコマンドを実行できる権利である。なおｍａｎａｇｅ権にはｗｒｉｔｅ権や、ｒｅａｄ権も含まれる。

これに対して一般の開発者が持つことのできる権利がｗｒｉｔｅ権やｒｅａｄ権である。ｗｒｉｔｅ権はファイルや情報を修正するようなサブコマンドを実行できる権利であり、ｒｅａｄ権も含まれる。なお、ｒｅａｄ権はファイルや情報を読むためのサブコマンドのみを、実行できる権利である。

ＣＯＲＯＮＡはプロジェクトとモジュールを階層的に扱うが、アクセス権も階層的に設定される。つまり、プロジェクトや１つ１つのモジュールに対して、別々にアクセスの権利を設定することによって、きめ細かいファイルの保護を行う。従って、ある開発者はＡのモジュールにはアクセスできるが、Ｂのモジュールにはアクセスできない、というような管理も実現できる。

プロジェクトへのアクセス権は、権利を持つ開発者を、プロジェクト全体に対する情報ファイルに登録しておくことによって、判定される。同様に、各モジュールへのアクセス権は、権利を持つ開発者を、それぞれのモジュールに対する情報ファイルに登録しておくことによって、判定される。これらの情報ファイルは、プロジェクトや各モジュールの下にあるｍａｎａｇｅディレクトリにおく。

４．６　開発者間のコミュニケーション

ＣＯＲＯＮＡの基本機能は、あるモジュールの修正による影響を開発者に知らせることであるが、修正を行った本人だけでなく、その影響が及ぶ先のモジュールを担当する開発者にもその情報を知らせる必要がある。しかし、このような作業は案外めんどうなものである。そこで、修正の影響が他のモジュールにも及ぶ場合、影響を受けるすべてのモジュールに対して、どのような影響があるかなどといった情報を自動的に送る機能を用意する。

これらの情報は、修正の影響を受けるそれぞれのモジュールのｉｎｆｏｒディレクトリにファイルの形で保存され、そのモジュールの担当者がデータベースにアクセスした際に、その旨を伝える。これをｉｎｆｏｒｍａｔｉｏｎ機能と呼ぶ。

また、ＵＮＩＸのｍａｉｌのように、開発者間で情報をやりとりする場合も考えられる。こちらは、先ほどのｉｎｆｏｒｍａｔｉｏｎ機能と違い、開発者の意志によって送られるもので、送る対象もモジュールではなく、個人である。また、送る内容や書式もまったく自由である。これをサポートするのがｍｅｓｓａｇｅ機能であり、この機能はＵＮＩＸのｍａｉｌ機能とほとんど同じであると考えてよい。

送られたメッセージは、やはりファイルの形で保存するが、こちらはプロジェクトの下のｉｎｆｏｒディレクトリに保存しておき、対象になる開発者がデータベースにアクセスした際に知らせる。

５．ＣＯＲＯＮＡの実現

開発管理システムＣＯＲＯＮＡはＶＡＸ－１１／７８５のＵＮＩＸ４．３ＢＳＤ上に実現するが、その実現にあたっては、いくつか考えなければならない課題がある。

５．１　操作性

まず、ＣＯＲＯＮＡにはいくつもの機能をもたせる。基本的な機能としては、データベースからソースコードを生成する機能。逆に、データベースにソースコードを登録する機能。また、コミュニケーションをサポートするｉｎｆｏｒｍａｔｉｏｎ機能や、ｍｅｓｓａｇｅ機能などもある。その他、開発者がソースコードを修正する際に、データベースから必要な情報を取り出すための機能なども考えなければならない。

このようにＣＯＲＯＮＡはいくつも機能を持つが、これらの機能をどのような形で実現するかが問題である。ＵＮＩＸ上の代表的な開発支援ツールであるＳＣＣＳやＲＣＳはどちらも、いくつかの機能をそれぞれ別々のコマンドとして用意している。つまり、ＳＣＣＳやＲＣＳとはコマンドの名前ではなく、コマンドの集まりとしてのシステムの名前である。このような形は、目的の作業を効率よく行うことができ、慣れれば使いやすいものであろう。しかし、行う作業ごとにコマンドの名前などを覚えなければならない。

そこで、ＣＯＲＯＮＡではそれ自身を大きなコマンドとして、その内部で対話的にサブコマンドを実行するという形をとる。つまり、ＣＯＲＯＮＡを起動すると、サブコマンドを待つ状態になるわけである。こうすることによって、サブコマンドにわかりやすい名前をつけることができ、使いやすいものになるだろう。しかし、単純な作業だけを行いたいのに、大きなコマンドであるＣＯＲＯＮＡを起動しなければならないという問題はある。

機能の実現の方法だけでなく、その対象の指定の仕方についても考える必要がある。ＣＯＲＯＮＡが行う作業はなんらかの形でデータベースにアクセスするのだが、そのアクセスの対象をデータベースの中から指定しなければならない。例えば、ソースコードを生成するにしても、データベース内にあるどのモジュールのソースコードを生成するかを指定する。

ＣＯＲＯＮＡをひとつのコマンドとすることによって、起動時には対象となるプロジェクト、つまりデータベースを指定しさえすればよいことになる。データベース内のどのモジュールを作業の対象にするのかは、その後サブコマンドを実行する際に指定すればよい。

しかし、あるひとつのモジュールに注目して作業を行っているのに、サブコマンドを実行するたびに、対象のモジュールを指定するのはやっかいである。そこで、まず対象のモジュールを決めてから、それに対するサブコマンドを実行する。従って、対象のモジュールを変えるまでは、サブコマンドの対象はすべて、先に設定したモジュールになる。このように現在注目しているモジュールを、カレントモジュールと呼ぶ。

これは操作性の向上につながるだけでなく、モジュールへのアクセス権の判定を行う上でも有益である。すでに述べたが、開発者の持つアクセスの権利によって実行できるサブコマンドは変わってくる。もしも、サブコマンドを実行する際にモジュールを指定するとしたら、そのモジュールへのアクセス権をサブコマンドを実行しようとするたびに調べなければならない。しかし、まず対象のモジュールを設定するのであれば、その際に一度だけアクセス権の判定を行えばよい。また、そのモジュールに対して開発者が持っているアクセス権の種類によって、実行できるサブコマンドをあらかじめ知ることができる。

また、サブコマンドの対象はモジュールだけではない。データベースに新しいモジュールを追加するような作業は、モジュールではなく、プロジェクトが対象であると考えられる。従って、プロジェクトに対して作業を行う場合は、対象をプロジェクトに設定する。

５．２　モジュール間の関係

あるモジュールに変更があった場合、その影響が他のどのモジュールに及ぶかを判断するには、各モジュール内にあるサブモジュールの情報を利用する。修正の影響が及ぶモジュールを効率よく判断するには、情報テーブル内にどのような情報を持てばよいかを考える。

例えば、ｍｏｄｕｌｅ１の中の変数ｘの定義が削除されたと仮定する。これはデータ定義サブモジュールの変更である。この影響は変数ｘを輸入している他のすべてのモジュールに影響を及ぼす。

他のモジュールのうちで、この影響を受けるのはどのモジュールであるのかは、他のモジュールのデータ宣言サブモジュールを調べればわかる。

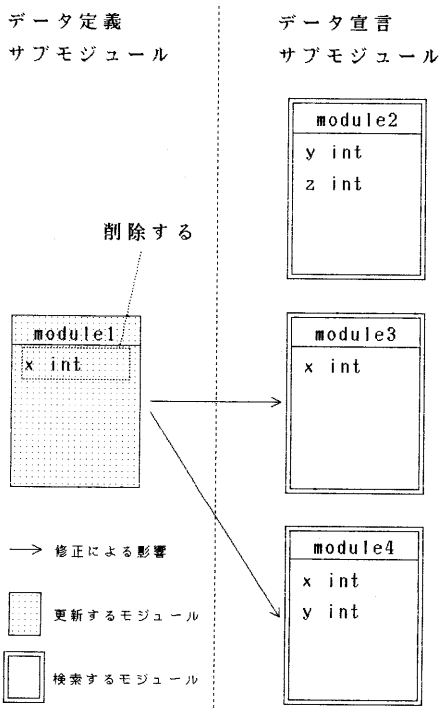

図７　関係の情報を持たない場合

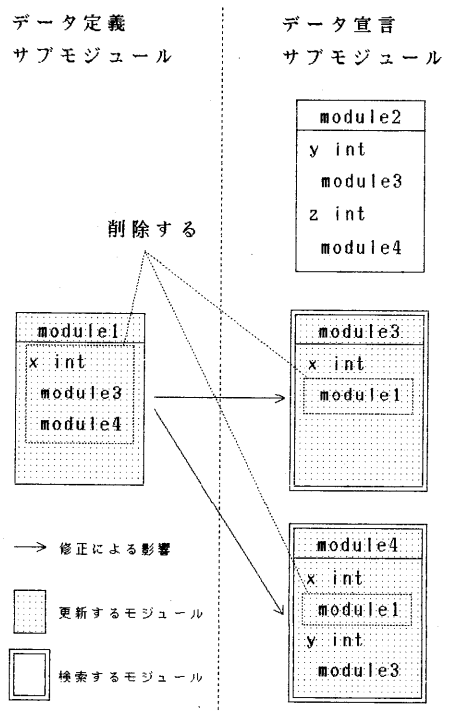

図８　関係の情報を持つ場合

まず、データ定義サブモジュールの情報テーブルに、輸出変数の型などの情報は持っているが、その変数をどこへ輸出しているのかという情報は持っていない場合を考える。同様に、データ宣言サブモジュールの情報テーブルにも、輸入変数の型などの情報はあるが、どこから輸入しているのかという情報はない（図７）。

この場合、変数ｘを輸入しているモジュールを知るには、他のすべてのモジュール（ｍｏｄｕｌｅ２～ｍｏｄｕｌｅ４）のデータ宣言サブモジュール情報テーブルを検索する必要がある。ｍｏｄｕｌｅ２は変数ｘに関しては、ｍｏｄｕｌｅ１と関係がないのに、その情報がないため、むだな検索をすることになる。従って、検索に時間がかかってしまい、決して効率がよいとは言えない。

しかし、データ宣言サブモジュール情報テーブルには、どのモジュールから変数ｘを輸入しているかという情報はもともと持っていないわけだから、変数ｘの輸入先がなくなったとしても、情報テーブルを書き換える必要はない。つまり、検索には時間がかかるが、情報テーブルの修正は、変更されたモジュール内のものだけですむ。

これに対して、データ定義サブモジュールの情報テーブルに、輸出変数の型などの情報と同時に、その変数をどこへ輸出しているのかという情報を持っている場合を考える。同様に、データ宣言サブモジュールの情報テーブルにも、輸入変数の型などの情報とともに、どこから輸入しているのかという情報が記述されている（図８）。

この場合は、変数ｘを輸出している先がデータ定義サブモジュール情報テーブルに記述されているので、修正の影響が及ぶモジュールを容易に知ることができる。従って、この情報を利用すれば、他のすべてのモジュールのデータ宣言サブモジュール情報テーブルを調べる必要はなくなる。変数ｘに関係のないｍｏｄｕｌｅ２は最初から検索しなくてよいのである。従って、輸出先として記述されているモジュールのデータ宣言サブモジュール情報テーブルだけを調べればよいから、むだな検索を避けることができ、検索にかかる時間が少なくてすむ。

しかし、変数ｘを輸入しているモジュールのデータ宣言サブモジュール情報テーブルには、どのモジュールから変数ｘを輸入しているかが記述されているから、その輸入先がなくなったいま、その記述を変更する必要がある。つまり、変数ｘを輸入しているすべてのモジュールのデータ宣言サブモジュール情報テーブルから、輸入先はｍｏｄｕｌｅ１であるという記述の部分を削除しなければならない。従って、前述の方法と比べると、検索は速いが、情報テーブルの修正にかかる時間が問題となる。

ただし、どんな種類の変更においても、情報テーブルを修正しなければならないとは限らない。例えば、単に輸出変数の初期値が変わっただけであれば、その変数を輸入しているモジュールの情報テーブルにすでに記述されている情報を、新しく書き換える必要はない。

このように、単純にどちらの方法がよいとは言えないが、今後、検討を加えたり、テストを行って決定する。

６．おわりに

本論文では、Ｃ言語を開発に用いる際に起こるさまざまな問題を解決する方法や、それをサポートする開発管理システムＣＯＲＯＮＡについて述べた。特に、グループ開発において問題になるファイルの保護は、ＣＯＲＯＮＡで用いる方法によって、確実に行うことができるだろう。

Ｃ言語でのソフトウェアの開発に、ＣＯＲＯＮＡを導入することにより、開発効率があがるものと考える。

今後、具体的にＣＯＲＯＮＡに持たせる機能（サブコマンド）を決定し、さらにＣＯＲＯＮＡが調べることのできる影響の範囲や種類を絞り決定する必要がある。ユーザインターフェイスについても、実際にユーザがＣＯＲＯＮＡを使ったときに、使いやすく、わかりやすいように設計するつもりである。従って、多くの意見や希望を出してもらい検討しなければならない。

また、先ほど述べた実現に関する課題や、データベースの規模、システムの大きさなどについても、検討が必要である。

参考文献

[1] B.W.Kernighan & D.M.Ritchie: "The C Programing Language"; Prentice-Hall (1978).

[2] Charles W.Krueger: "The SMILE Reference Manual" The GANDALF System Reference Manuals (1986).

[3] Dennis.M.Ritchie: "The C Programing Language - Reference Manual" 4.3BSD UNIX Programmer's Supplementary Documents, Vol.1 (1986).

[4] Eric Allman: "An Introduction to the Source Code Control System" 4.3BSD UNIX Programmer's Supplementary Documents, Vol.1 (1986).

[5] 村井　純，井上尚司，砂原秀樹："プロフェショナルＵＮＩＸ"； アスキー出版局 (1986).

[6] 西田親生，五月女健治："Ｃ言語プログラミング・ツール"； 啓学出版 (1986).

[7] 坂本　文："ＵＮＩＸツール ガイドブック"； 共立出版 (1986).

[8] Walter F.Tichy: "An Introduction to the Revision Control System" 4.3BSD UNIX Programmer's Supplementary Documents, Vol.1 (1986).