



Enterprise Architect making sequence diagram feature guide

by SparxSystems Japan

Enterprise Architect シーケンス図の作成 機能ガイド

(2018/09/14 最終更新)



このドキュメントでは、デバッグ機能から派生して利用できるシーケンス図の作成機能について紹介します。

なお、このドキュメントでは、Enterprise Architect 14.1 ビルド 1427 を利用しています。

1 概要

Enterprise Architect のシーケンス図の作成機能は、基本的に Enterprise Architect のデバッグ機能を実行中に得られる情報を元に作成します。そのため、Enterprise Architect 内部でプログラムを動作させてデバッグ中の情報を取得する必要があります。

つまり、ソースコードを直接解析してシーケンス図を生成する方法ではありません。よって、生成したシーケンス図には、ソースコードに存在する「if 文」のような分岐は含まれません。実際に動作した結果のみが表示されます。if 文のような分岐がある場合に両方のシーケンス図を得たい場合には、プログラムを異なる条件で動作させ、複数のシーケンス図を生成する必要があります。

なお、Enterprise Architect の機能として、ソースコードを直接解析する方法を希望されるお客様もいらっしゃいます。「処理が複雑で見てもよくわからないソースコードの解析のために、シーケンス図を可視化したい」という問題の解決のために、このシーケンス図の自動生成に興味を持つ方が多いです。

しかし、もし仮にソースコードの内容を解析し、そのまま可視化とした場合には、得られるシーケンス図もやはり複雑で、見てもよくわからないものになります。ソースコードをそのまま解析し可視化することは、ほとんどの場合には解決策になりません。(汚いソースコードがきれいなシーケンス図として見えることを期待される方が多いですが、汚いソースコードの内容をそのまま図にすれば、汚いシーケンス図しか得られません。)

そのため、可読性もあり実際に有用な結果が得られる、デバッグ実行からのシーケンス図生成という方法を Enterprise Architect では採用しています。

Enterprise Architect のデバッグは、Enterprise Architect が独自に開発し保持するデバッグ機能を利用します。デバッグを行う環境に Visual Studio 等のデバッグ環境のインスト

ールは不要です。

ただし、いずれの言語・環境においても、**Enterprise Architect** はビルド環境を提供していませんので、デバッグ対象の(デバッグ情報を含む)実行ファイルを別途作成する必要があります。例えば、**C#**, **VB.NET** および **Visual C++** の場合には、**Visual Studio** で作成したデバッグ版の実行ファイルが必要となります。**Java** の場合には、デバッグ情報を含むファイルを作成する必要があります。

また、基本的にはビルド時のソースファイルの位置と、**Enterprise Architect** に読み込むソースファイルの位置が完全に一致していなければなりません。そのため、最初はデバッグを行うマシンでビルドし、そのマシン上でそのままソースファイルを **Enterprise Architect** に読み込んで、デバッグ機能を実行することをお勧めします。(ソースファイルのパスが一致しない場合には設定したブレークポイントで停止せず、シーケンス図の自動生成機能も利用できません。)

このドキュメントは、**Enterprise Architect** のシーケンス図の自動生成機能の概要を紹介するためのものです。特定の言語・環境での生成方法の詳細を記載したものではありません。必要に応じて、ヘルプファイルもご覧下さい。

2 必要条件

この機能を利用するには、対象の言語ごとにいくつかの条件があります。なお、言語ごとの条件以外にも、以下のような制限があります。

- ・ アンチウイルスソフトによってデバッグの動作が制限される場合があります。もし動作しない場合、アンチウイルスソフトを無効にして動作するかどうかご確認ください。
- ・ 利用する言語(デバッガ)によっては、実行ファイルなど、動作解析の設定として入力する内容にパス情報に全角文字列を含むと正常に動作しない場合があります。このような場合には、全角文字を含まないようなパスにソースコードを配置し、ビルド・ソースコードの読み込み後、シーケンス図の作成を実行して下さい。
- ・ この機能を利用することで、アプリケーションの動作速度にも影響します。そのため、通常の動作では問題ないアプリケーションでも、機能を実行すると動作が変わったり、不正終了してしまうという状況もあるようです。このような問題が発生す

る場合には、元のアプリケーションを修正する以外に対処方法はありません。

- さまざまな問題が発生した場合に、Enterprise Architect のデバッグサブウィンドウに問題の解決の参考になる情報が表示される場合があります。

2.1 Java の場合

Java を利用する場合には、JDK5.0(旧 Java1.5)以降が必要になります。デバッグ情報を含むクラスファイルが対象です。

なお、JDWP (Java Debug Wire Protocol)を利用する場合には、ローカル・リモート環境にある Java アプリケーションや Android アプリケーション(実機およびエミュレータ)でも利用できます。

2.2 .NET の場合

.NET アプリケーション(C#, Visual C++, VB.NET)の場合には、デバッグ版としてビルドしたアプリケーションが対象です。

2.3 C++(アンマネージ・コード)の場合

.NET を利用していないアプリケーションの環境を、この機能ガイドや Enterprise Architect の画面・ヘルプファイルなどでは「Windows Native」と表現します。

Visual Studio で作成した、.NET ではない C++の場合には、Visual Studio でデバッグ版としてビルドしたファイルのみがこの機能の対象になります。Windows 環境であっても、Visual Studio 以外のコンパイラで作成したデバッグ版は利用できません。(Visual Studio が作成するデバッグ情報ファイルを参照して、デバッグ中の位置を特定します。)

2.4 PHP の場合

PHP のバージョン 5.3 以降が必要です。また、デバッグのために PHP zend extension XDebug 2.1 以降を利用します。

2.5 GDB の場合

gdb.exe と Enterprise Architect が通信することで、C 言語や C++言語で作成されたアプリケーションの動作解析ができます。gcc や g++でコンパイルする際に、デバッグ版として

デバッグ情報を含む必要があります。リモートの **GDB Server** と通信して、他のマシン上のアプリケーションを対象にすることもできます。古いバージョンの **GDB** では正常に動作しない場合があります。

3 注意点

このシーケンス図の自動生成機能を利用する場合、注意点があります。

3.1 作成する範囲について

このシーケンス図の自動生成機能は、対象のアプリケーションの**全ての動作内容を最初から最後まで取得する使い方は想定していません**。その結果表示されるシーケンス図は、人間が見て解読できないものになることが多く、またシーケンス図が大きくなりすぎて可読性が低いことが、その理由です。また、解析結果が膨大になることが多く、シーケンス図の作成処理に多くの時間とメモリを必要とします。また、作成した内容を表示する際や表示内容をスクロールする際に、シーケンス図が非常に大きいため、マシンの性能によっては長い時間がかかることがあります。いずれにしても、非常に大きなシーケンス図は実用的ではなく、またツールとしても非常に大きなシーケンス図が生成されるような使い方は想定していません。

そのため、解析したい範囲のみを対象にして、シーケンス図を作成します。あるいは、フィルタの機能で、関係のない部分を記録しないようにする方法もあります。フィルタの機能については、ヘルプファイルの「フィルタの設定」をご覧ください。

3.2 オブジェクト指向ではない言語について

このデバッグ機能は、**Visual Basic 6.0・C 言語 (Visual Studio でコンパイルしたもの)** についても実行可能ですが、現時点では適切な結果にはなりません。
(gcc でコンパイルした C 言語の場合には、1 つのライフラインが 1 つのファイルとして、ファイル間の関数呼び出しの経緯がシーケンス図として表現されます。)

4 Visual Studio のアプリケーションの場合の実行までの手順

Enterprise Architect バージョン 14 で追加された機能を利用すると、**Visual Studio** で作成されたアプリケーションのシーケンス図の自動生成を効率よく行うことができます。ここでは、その手順を紹介します。

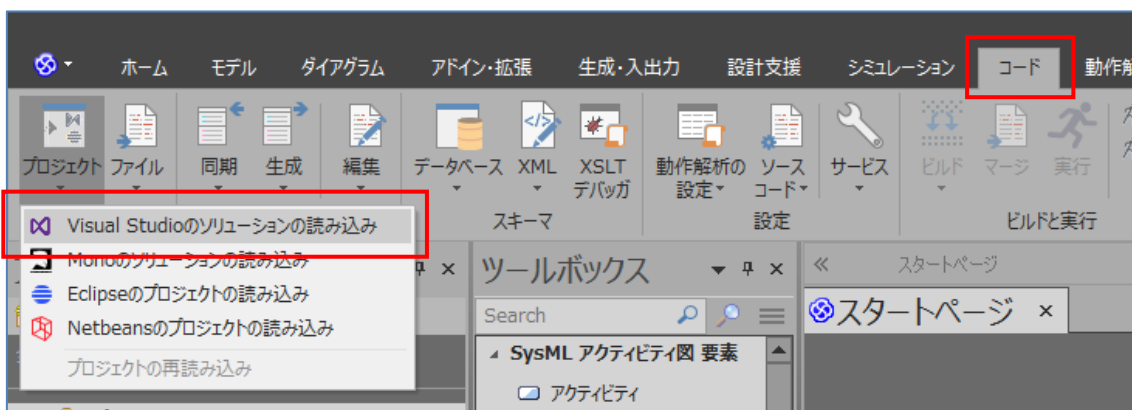
5.1. ソリューションの読み込み

まず、新規に空のプロジェクトを作成し、プロジェクトブラウザでパッケージを 1 つ追加してください。ここまでの具体的な手順が不明な場合には、小冊子「ゼロからはじめる Enterprise Architect」の内容を一通り操作し、基本的な操作方法を体験してください。

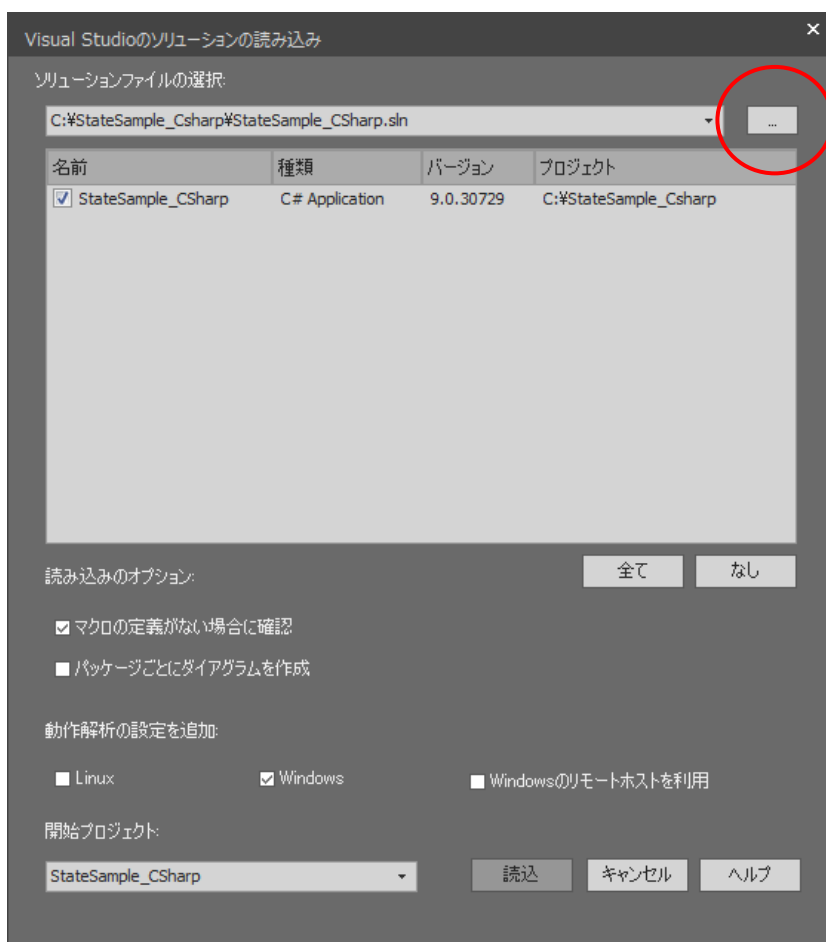
小冊子「ゼロからはじめる Enterprise Architect」について

https://www.sparxsystems.jp/products/EA/ea_zero.htm

作成したパッケージをプロジェクトブラウザ内で選択した状態で、「コード」リボンの左端にある「読み込み」パネル内の「プロジェクト」ボタンを押してください。メニューが表示されますので、「Visual Studio ソリューションの読み込み」を選択します。



「Visual Studio ソリューションの読み込み」画面が表示されますので、「ソリューションファイルの選択」の入力欄の右端にある「...」(参照)ボタンを押し、拡張子 sln の Visual Studio のソリューションファイルを指定してください。ソリューションに含まれるプロジェクトが一覧に表示されます。



「パッケージごとにダイアグラムを作成」にチェックを入れると、パッケージ単位でクラス図を自動生成します。また、「動作解析の設定を追加」の項目で「Windows」にチェックが入っていることを確認してください。

準備が整ったら「読み込」を実行してください。途中、エラーが表示される場合には、ドキュメント「ソースコードの生成と読み込み 機能ガイド」やヘルプの内容をご覧ください。

ドキュメント「ソースコードの生成と読み込み 機能ガイド」のダウンロード

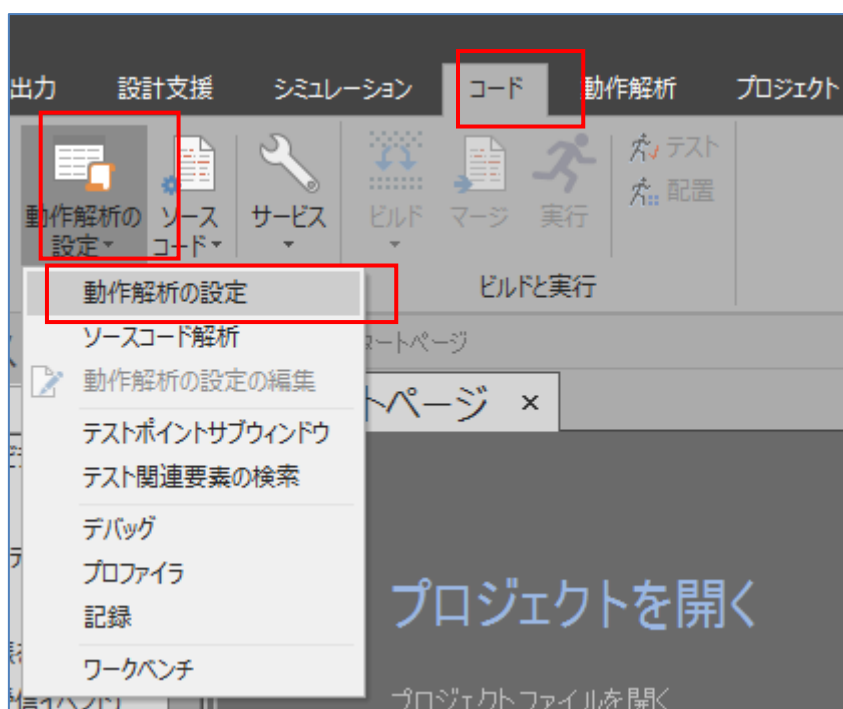
https://www.sparxsystems.jp/products/EA/ea_documents.htm

5.2. 動作解析の設定の修正

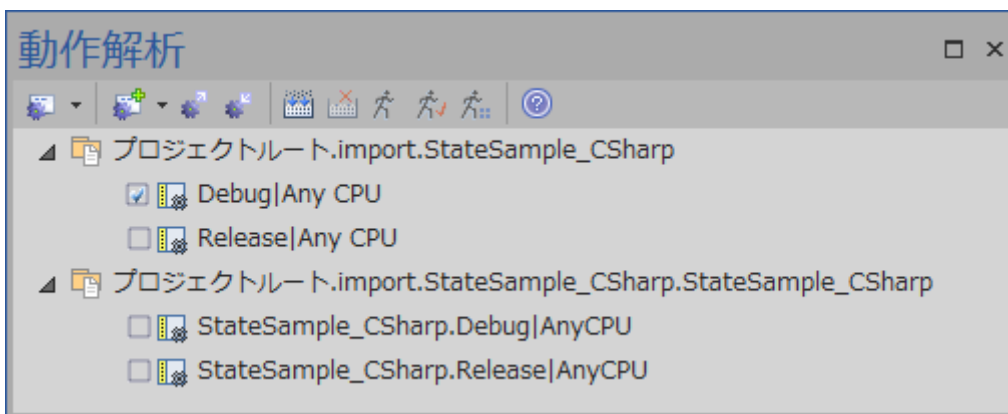
シーケンス図の自動生成機能の実現には、Enterprise Architect の「デバッグ」の機能を利用しています。このデバッグの機能を利用するためには、「動作解析の設定」が必要となります。

「動作解析の設定」の基本的な内容は、先の手順で自動的に生成されていますが、現時点では常に 64 ビット OS 用の設定となります。そのため、32 ビット OS で実行する場合には、以下の修正が必要となります。

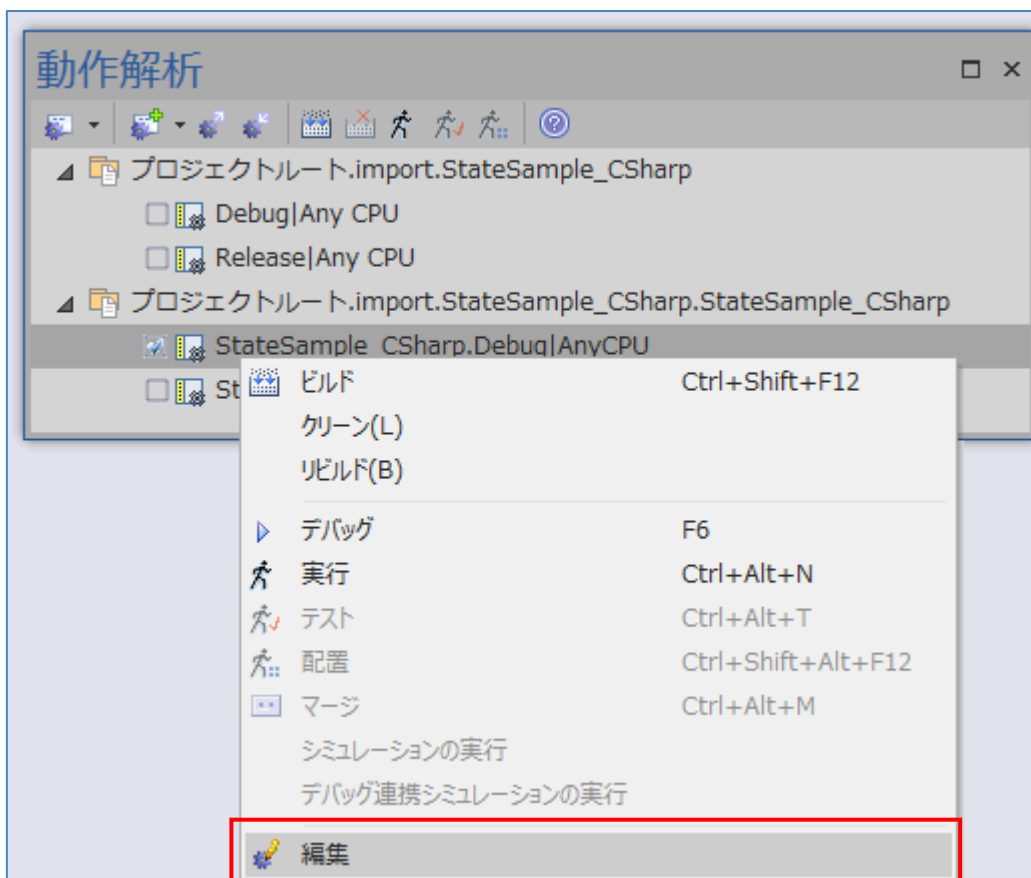
設定の変更には動作解析の設定サブウィンドウを利用します。「コード」リボン内の「設定」パネル内にある「動作解析の設定」ボタンを押し、表示されるメニューから「動作解析の設定」を選択してください。



動作解析の設定では、通常は次の画像のように、少なくとも 4 つの定義が追加されます。ソリューション全体の Debug/Release ビルドと、ソリューションに含まれるプロジェクトの Debug/Release ビルドに対応します。

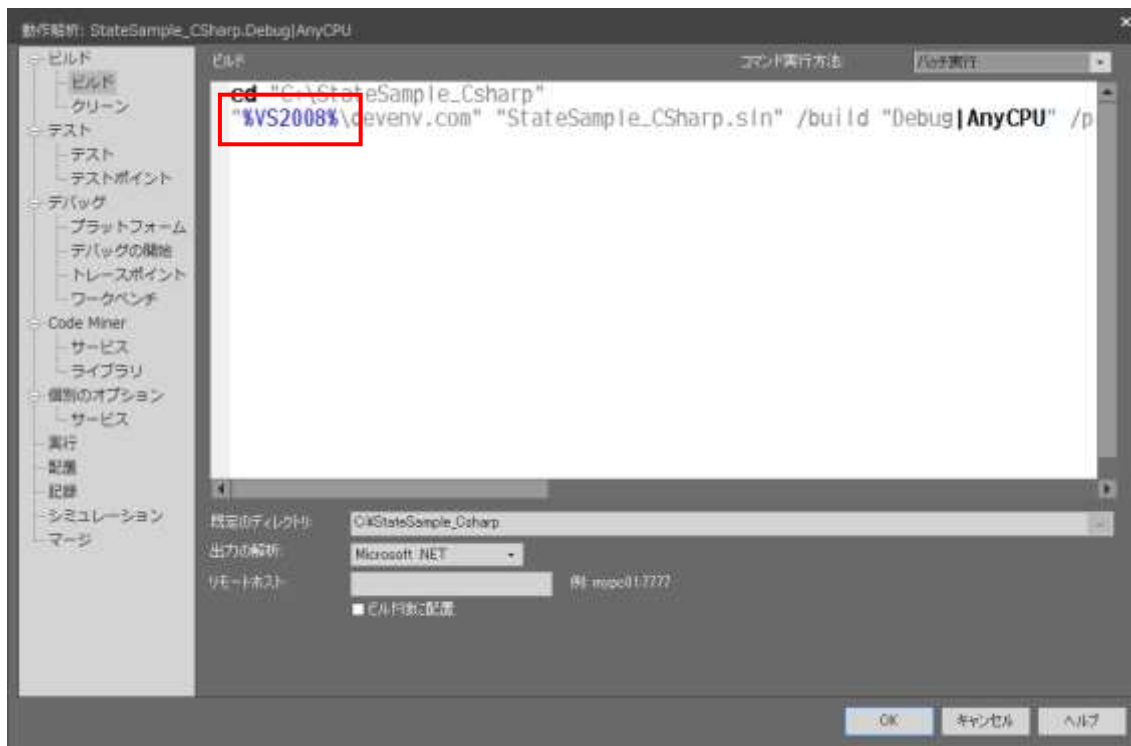


今回利用するのはプロジェクトに対応する **Debug** ビルドですので、該当する項目にチェックを入れ、さらに右クリックして「編集」を選択してください。「動作解析」画面が表示されます。

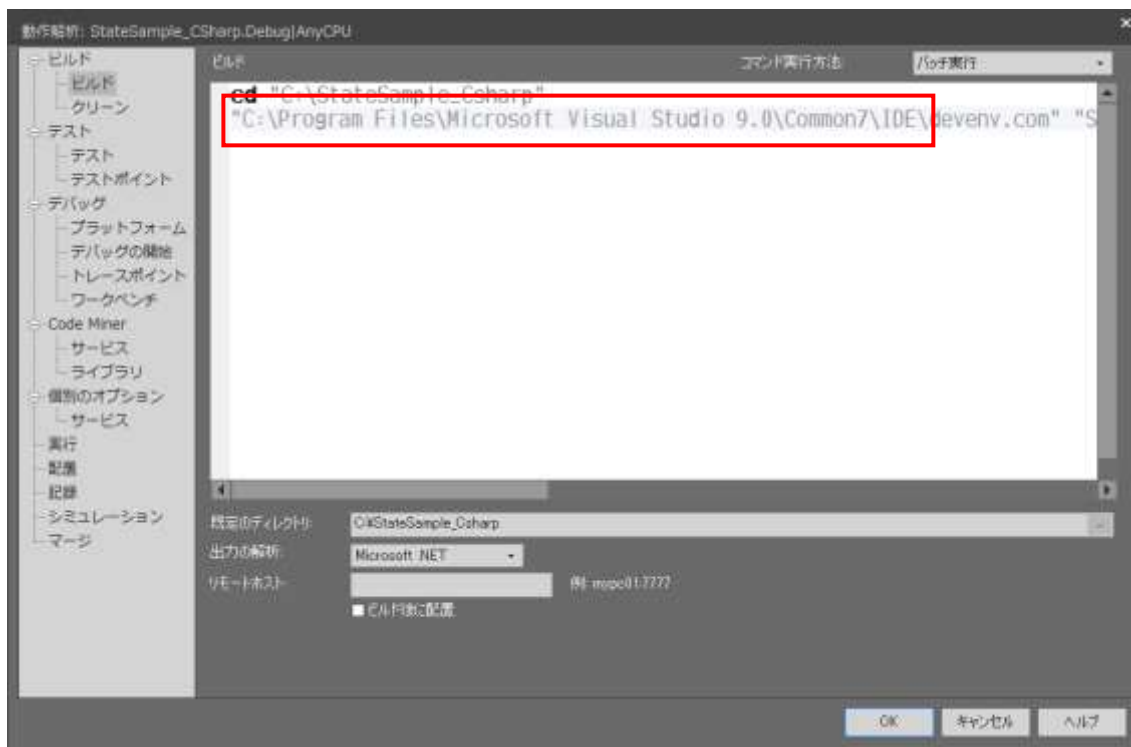


この設定中の「ビルド」および「クリーン」に含まれる定義内容には、「%VS2008%」や「%VS2013%」など、読み込んだ Visual Studio のバージョンと関係するマクロが利用され

ています。しかし、この値は 64 ビット環境の情報に展開されるため、32 ビット環境ではデバッグビルドの際にエラーとなってしまいます。



そこで、このマクロを削除し、実際のパスに置き換えてください。次の画像は、パスを置き換えた例です。



これで、動作解析の設定は完了です。

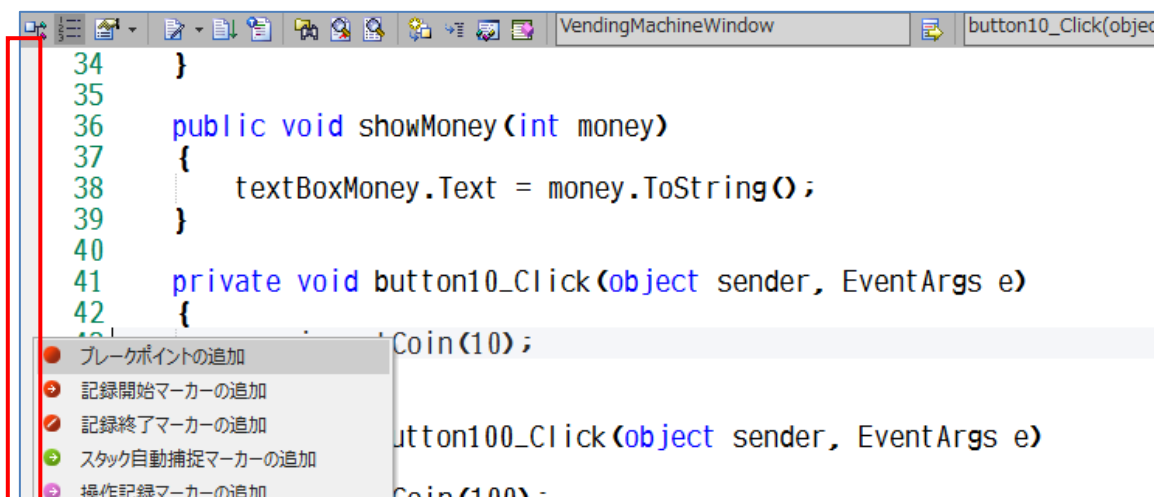
5.3. ブレークポイントの設定

次に、動作を記録する開始点となるブレークポイントを配置します。なお、3.1章で説明したように、main関数の先頭にブレークポイントを配置して全部の情報をシーケンス図に生成するなど対象が広すぎると、シーケンス図の作成に時間がかかるほか、図が大きすぎて意味がない図になることが多いです。まずは、動作を把握したい特定のメソッド(関数)のみを対象としてください。

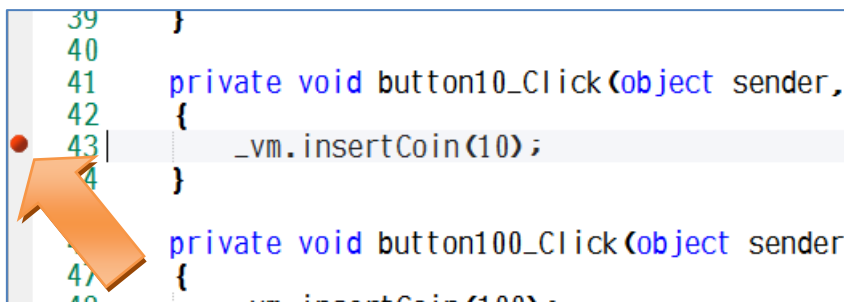
ブレークポイントを配置すると、対象のプログラムはその場所で動作を一時停止します。そのため、マルチスレッドのプログラムや多くのプログラムと連携して動作するようなプログラムでは、動作が変わり意図したとおりに動作しないこともあります。その場合には、6章で説明する「マーカー」を配置してください。

ブレークポイントを配置するには、クラス図で対象のクラス要素を選択し、F12キー(あるいは「コード」リボン内の「ソースコード」パネルにある「編集」ボタン)を押し、「タブで表示」を選択する)を実行して、Enterprise Architectのタブとしてソースコードを表示します。

ブレークポイントを設置する行の左端にある灰色の列(下の画像の赤枠の範囲)を右クリックするとブレークポイントを設定するためのメニューが表示されますので、「ブレークポイントの追加」を選択してください。該当の領域を左クリックで追加することもできます。



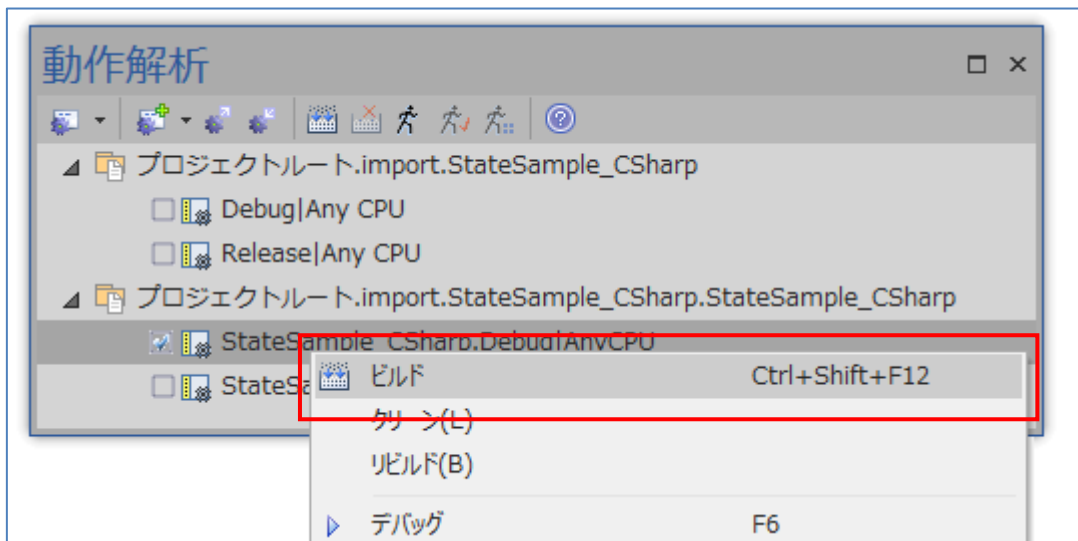
追加すると、以下のようにブレークポイントが明示されます。



5.4. 実行とシーケンス図の生成

設定が完了したら、デバッグ版としてのビルドを行います。動作解析の設定サブウィンドウで **Debug** ビルドの設定を右クリックし、「ビルド」を実行します。出力サブウィンドウが自動的に開き、ビルドの結果が表示されます。「~の処理は終了コード0で完了しました。」のように、終了コード0(0は正常終了を示す値)で終わっていることを確認してください。

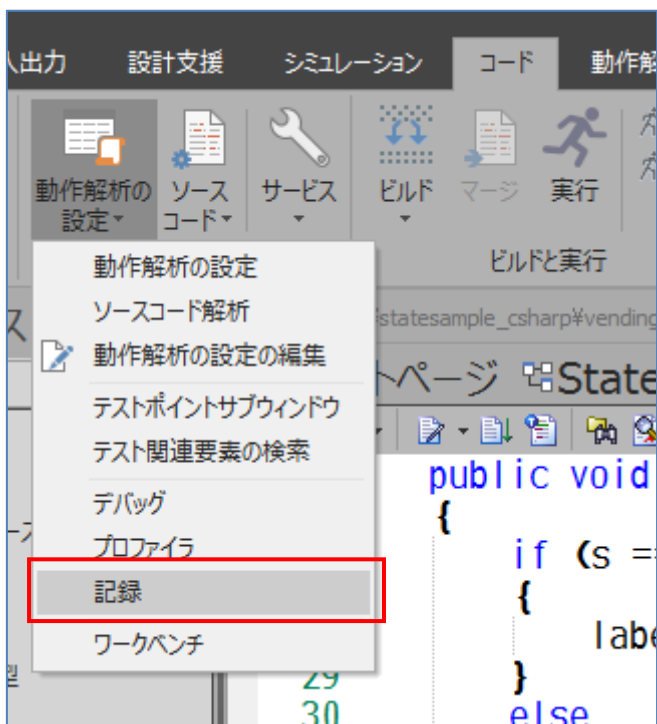
なお、動作解析の設定サブウィンドウを開くには、「コード」リボン内の「設定」パネル内にある「動作解析の設定」ボタンを押し、表示されるメニューから「動作解析の設定」を選択してください。



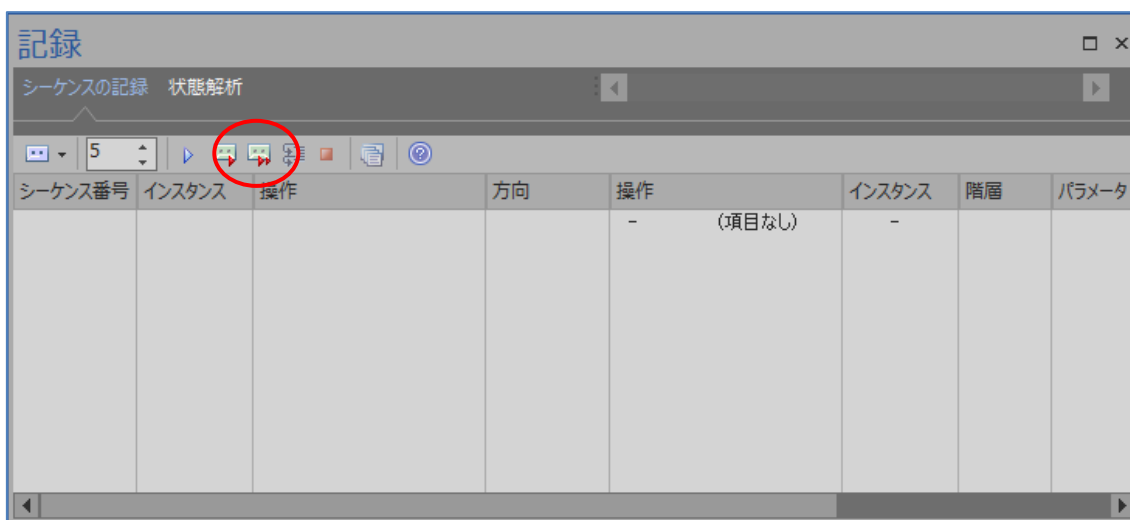
次に、先ほどと同じ右クリックメニューから「デバッグ」を実行してください。デバッグ版のプログラムが起動し、設定したブレークポイントの位置に達したら、ソースコード内で停止したことが明示されます。

```
39     }
40
41     private void button10_Click(object sender, EventArgs e)
42     {
43         _vm.InsertCoin(10);
44     }
45
46     private void button100_Click(object sender, EventArgs e)
47     {
```

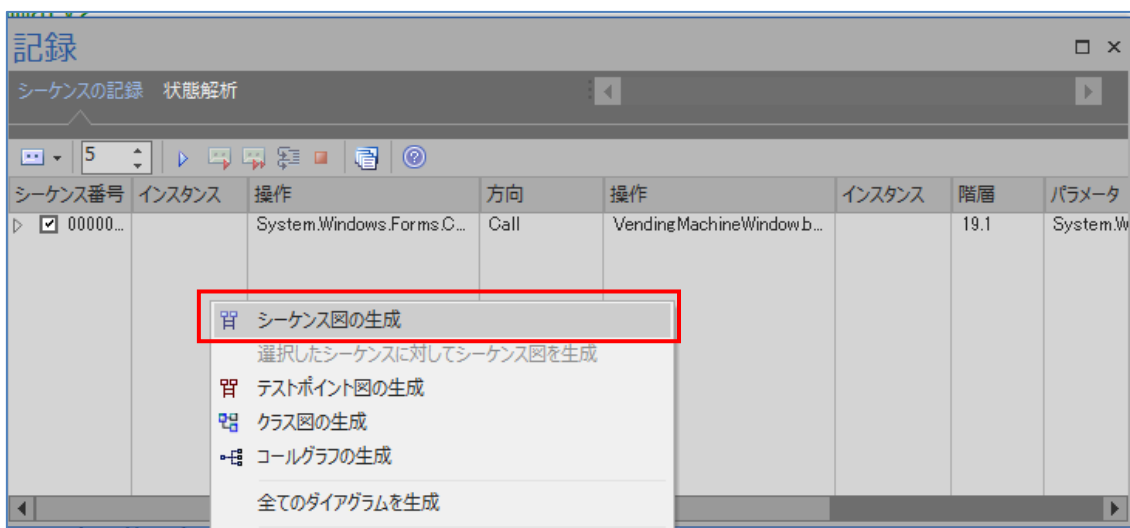
停止した位置からの処理内容を記録し、シーケンス図として生成するためには記録サブウィンドウを利用します。記録サブウィンドウは、動作解析の設定サブウィンドウを表示する際に利用したボタンと同じ、「コード」リボン内の「設定」パネル内にある「動作解析の設定」ボタンを押してください。表示されるメニューから「記録」を選択してください。



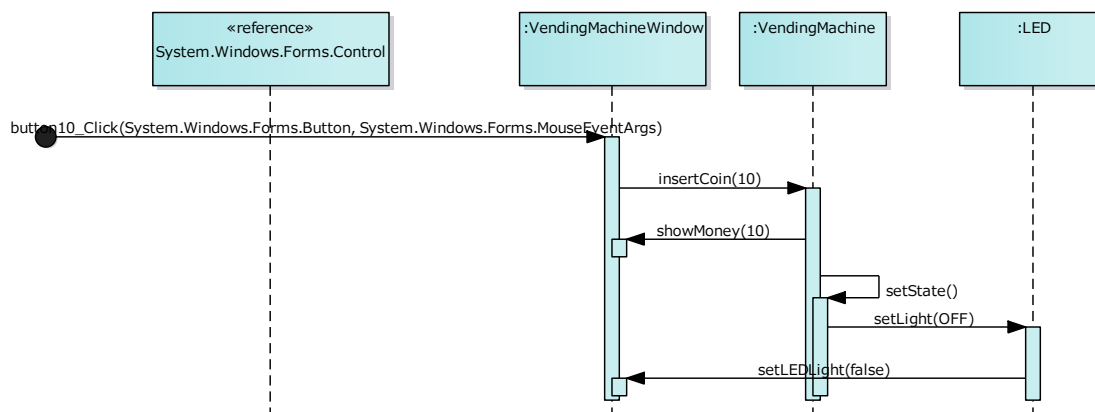
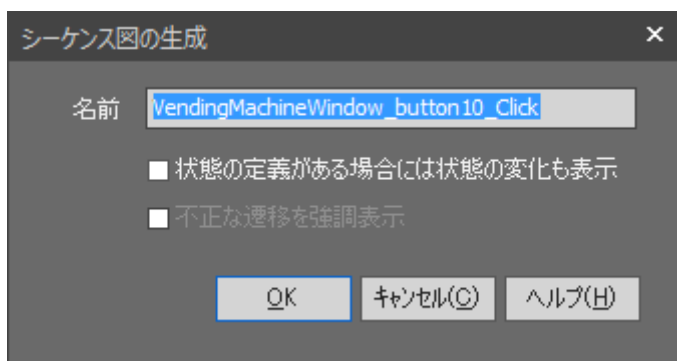
記録サブウィンドウが表示されますので、ツールバーの「選択したスレッドを自動的に記録」ボタン(下の画像の赤丸のボタン)を押します。停止していたデバッグ動作が再開され、記録サブウィンドウにログが表示されます。プログラムを終了させると、デバッグ実行も停止します。



記録された内容からシーケンス図を生成するには、記録サブウィンドウ内の一覧の背景を右クリックしてメニューを出し「シーケンス図の生成」を選択してください。



以下のような「シーケンス図の生成」画面が表示されますが、今回は特に設定を変更する必要はありませんので、そのまま「OK」ボタンを押してください。シーケンス図が生成され、表示されます。



5 一般的な言語での実行までの手順

ここでは、実際に環境を構築してシーケンス図を作成するまでの流れを説明します。また、設定の一例として **C#**および **Java** の例を紹介します。設定の流れはどの言語の場合でも同じです。**C#**と **Java** 以外の言語の設定については、ヘルプファイルをご覧ください。また、対象のアプリケーションにより、追加の設定が必要な場合もあります。

5.1. ソースファイルの読み込み

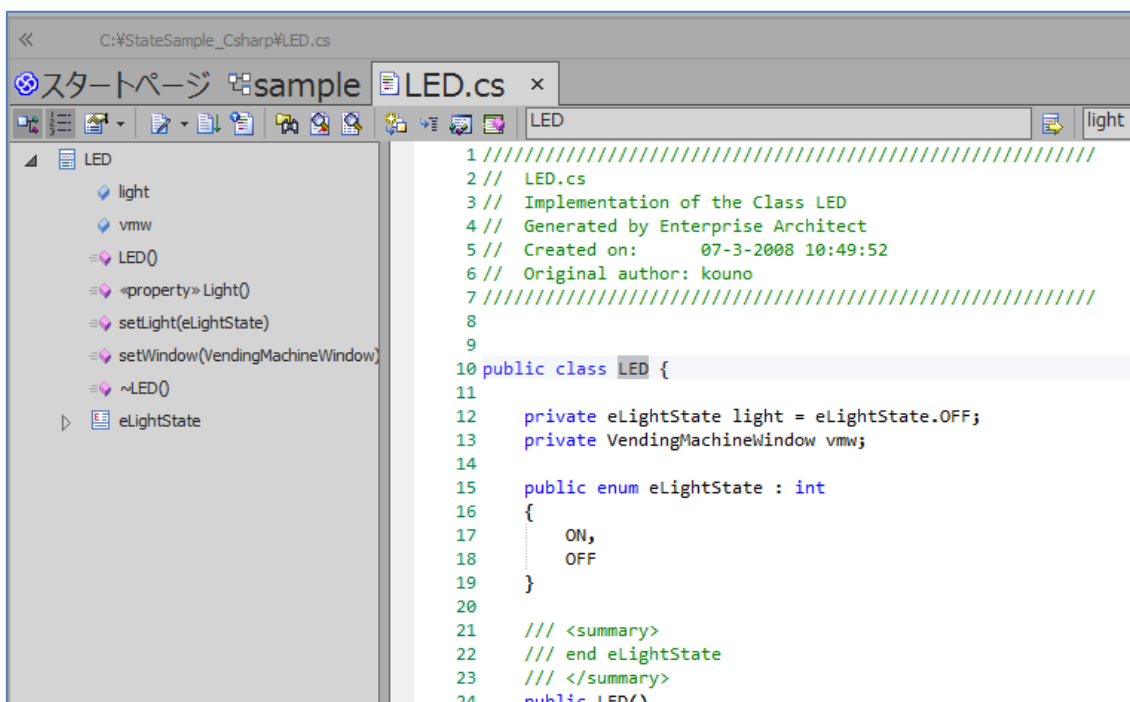
デバッグ・シーケンス図の自動生成を実行するための条件の一つとして、対象のソースコードがクラス図として表現されていることが必要です。そのため、まず、対象のソースファイルを **Enterprise Architect** のクラス図として読み込みます。

この詳細は、ヘルプファイルの他、ドキュメント「ソースコードの生成と読み込み 機能ガイド」でも説明しています。下記ドキュメントの内容をご覧ください、ソースファイルを読み込んでください。

ドキュメント「ソースコードの生成と読み込み 機能ガイド」のダウンロード

https://www.sparxsystems.jp/products/EA/ea_documents.htm

読み込んでクラス図が作成できたら、任意のクラス要素を選択した状態で **F12** キー(あるいは「コード」リボン内の「ソースコード」パネルにある「編集」ボタン)を押し、「タブで表示」を選択する)を実行して、**Enterprise Architect** のタブとしてソースコードが正しく表示されるかどうかを確認してください。

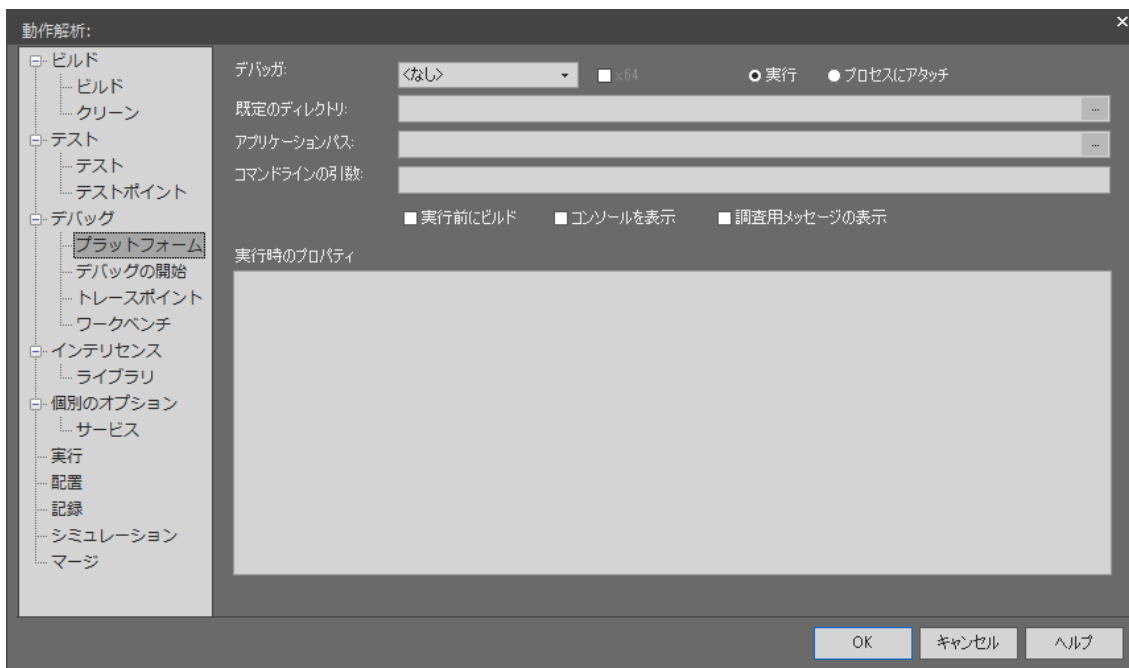
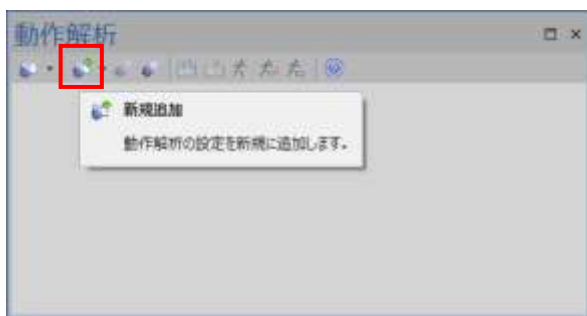


Enterprise Architect のデバッグの実行のためには、あらかじめ対象となるクラスのソースコードを、このような形でタブとして開くことができるように設定する(ソースコードの位置とクラス要素を正しく関連づける)必要があります。

5.2. ビルドとデバッグの設定

次に、先ほどソースファイルを読み込んだパッケージに対して、ビルドとデバッグの設定を行います。先ほど読み込むときに選択したパッケージをプロジェクトブラウザ内で選択した状態で、「動作解析」リボン内の「実行」パネルにある「動作解析の設定」ボタンを押してください。ソースファイルを読み込んだパッケージが階層構造になっている場合(名前空間やディレクトリに対応するパッケージが自動生成された場合)には、読み込みを実行した、最上位のパッケージに対して設定を行ってください。

「動作解析」サブウィンドウが表示されますので、ツールバーの左から2番目にある「新規追加」ボタンを押し、設定を追加してください。自動的に動作解析ダイアログが開きます。



この画面で今回入力する必要があるのは、デバッグ以下の「プラットフォーム」グループです。上の画面は、画面左側にあるツリー一覧で「プラットフォーム」を選択した状態です。

5.2.1. デバッグ

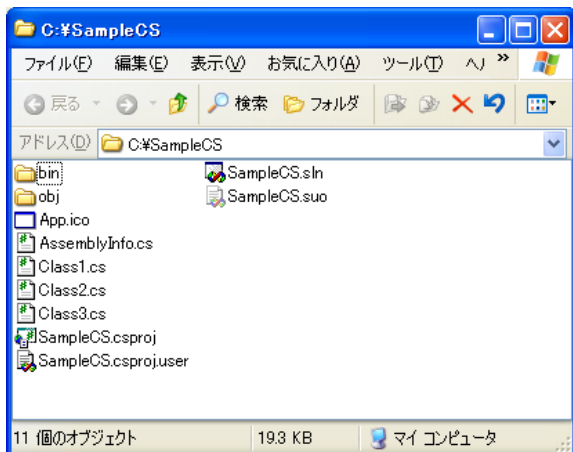
「デバッグ」の選択肢では、対象のアプリケーションの動作環境(言語)を選択してください。

5.2.2. 既定のディレクトリ

既定のディレクトリは、ソースファイルなどが置いてあるディレクトリになります。このサンプルでの指定例を画像でご紹介しますので、ご参考にしてください。

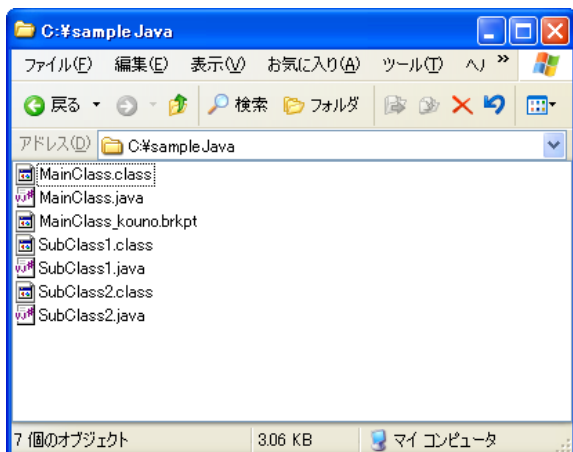
- C#の例

デバッガ:	Microsoft .NET	<input checked="" type="radio"/> 実行	<input type="radio"/> プロセスにアタッチ
既定のディレクトリ:	C:\sampleCS		



- Java の例

デバッガ:	Java	<input checked="" type="radio"/> 実行	<input type="radio"/> 仮想マシン(VM)にアタッチ
既定のディレクトリ:	C:\sampleJava		



5.2.3. デバッグ対象の指定

最後に、デバッグ対象の設定です。ここも、言語ごとに実際の例でご紹介します。

- C#・Windows Native

アプリケーションパス:	<input type="text" value="bin\Debug\sampleCS.exe"/>
コマンドラインの引数:	<input type="text"/>
	<input type="checkbox"/> 実行前にビルド <input type="checkbox"/> コンソールを表示 <input type="checkbox"/> 調査用メッセージの表示
シンボルの検索パス:	<input type="text"/>

- アプリケーションパスに、「既定のディレクトリ」欄で指定した位置からの相対パスでの実行ファイルを指定してください。

○ Java

アプリケーションクラス名:	<input type="text" value="Customer"/>
コマンドラインの引数:	<input type="text"/>
	<input type="checkbox"/> 実行前にビルド <input type="checkbox"/> コンソールを表示 <input type="checkbox"/> 調査用メッセージの表示
Java仮想マシンのオプション:	<input type="text" value="jre=C:\Program Files\Java\jdk.1.6.0_23;-Djava.class.path=%classpath%O:\sampleJava"/>

- 実行およびデバッグの欄に、デバッグを開始する対象のクラスの名前を入力してください。
- 引数には、ランタイムの情報を指定します。詳細はヘルプファイルをご覧ください。

この設定が正しくない場合、デバッグを開始した場合に何も反応がないか、あるいはエラーが表示されます。エラーの内容は、デバッグサブウィンドウに表示されます。デバッグサブウィンドウは、「動作解析」リボン内の「分析」パネルにある「デバッグ」ボタンを押すと表示されるメニューから「デバッグサブウィンドウ」を選択すると表示できます。うまく動作しない場合には、このウィンドウに表示される内容を確認して下さい。解決のヒントになる情報が表示されている場合があります。

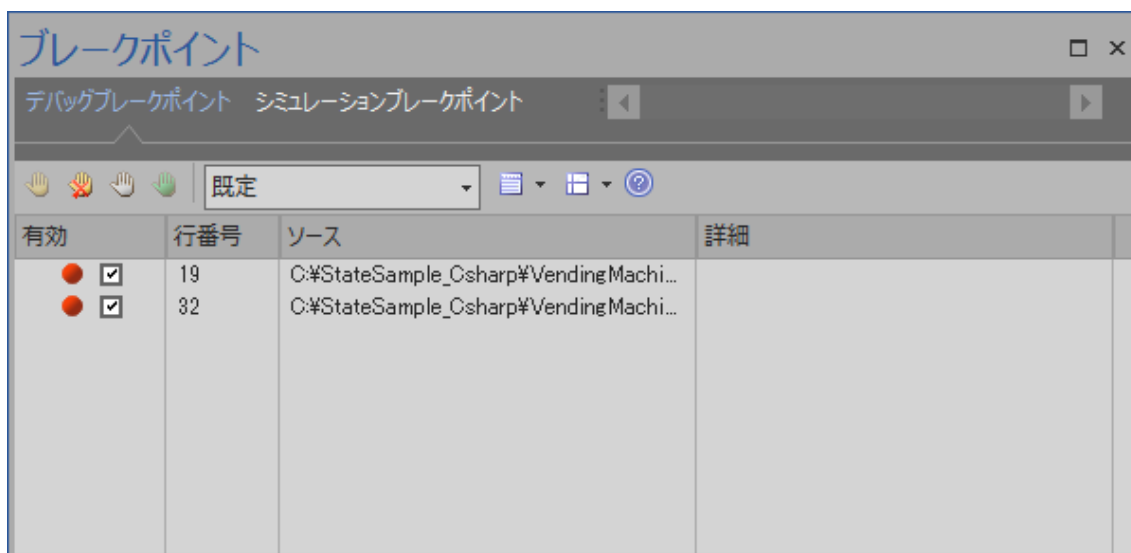
以上で設定は完了です。

5.3. デバッグの実行とシーケンス図の作成例

最後に、Enterprise Architect でデバッグをしてシーケンス図を作成するための情報を収集する方法をご紹介します。

なお、このデバッグの機能は、Enterprise Architect 内で全て指定・実行します。Visual Studio など他の IDE(統合開発環境)内で設定されているブレークポイント情報などは関係なく、利用することもできませんのでご注意ください。

まず、ブレークポイントサブウィンドウを表示してください。このサブウィンドウは、「動作解析」リボン内の「ウィンドウ」パネルにある「ブレークポイント」ボタンを押すと表示されます。



これから、シーケンス図を作る範囲を指定します。この方法として、ブレークポイントあるいはマーカーを利用します。マーカーについては後述します。

該当のクラスのソースコードを(F12 キーで)Enterprise Architect 内のタブとして表示し、ブレークポイントを設定する行の左端の余白を左クリックします。下の例で、赤丸が表示されている列を左クリックするとブレークポイントを設定することができます。シーケンス図を作成開始したい位置に設定してください。

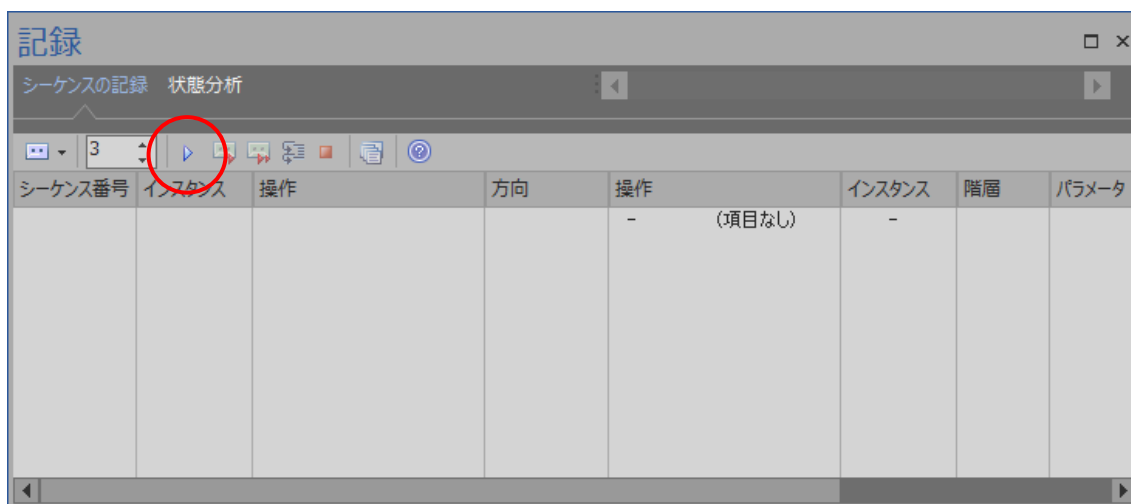
```

12
13 public class Customer
14 {
15
16     public static void Main()
17     {
18         VendingMachine vm = new VendingMachine();
19
20         vm.insertCoin(100);
21
22         for (int i = 0; i < 2; i++)
--

```

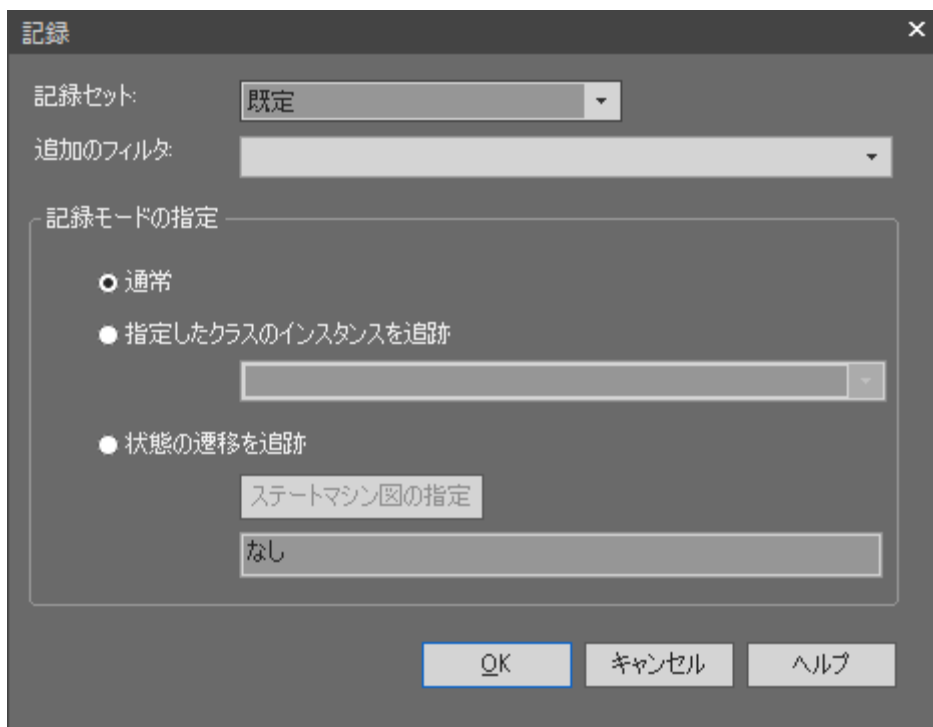
シーケンス図の作成のためのデバッグを実行するには、記録サブウィンドウを利用します。「動作解析」リボン内の「分析」パネルにある「記録」ボタンを押すと表示される「記録サブウィンドウ」を実行すると表示されます。

この状態で、このサブウィンドウ上部のツールボックスの左から 3 番目にある「記録」(青矢印)ボタンを押します。



ボタンを押すと、以下のような画面が表示されますが、ここでは何も設定せず、そのまま OK ボタンを押してください。

(ステートマシンとの連携などを行う場合には、設定が必要です。)



すると、設定が正しく行われている場合、ブレークポイントを設定した場所にカーソルがあたり、デバッグモードになります。(下の例をご覧ください。)

```

1  \
2  \
3  \
4  \
5  \
6  \
7  \
8  /// <summary>
9  /// end machineState
10 /// </summary>
11 private LED led;
12 private eMachineState state;
13 private int totalAmount = 0;
14 private VendingMachineWindow vmw;
15
16 public static void Main()
17 {
18     VendingMachine vm = new VendingMachine();
19     vm.showWindow();
20 }
21
22 private enum eMachineState

```

このときに、デバッグに関するさまざまなサブウィンドウを利用している場合には、それぞれのサブウィンドウには情報が表示されます。また、ブレークポイントで停止しない場合など、問題がある場合には、「デバッグ」サブウィンドウにメッセージが表示されている場合があります。デバッグサブウィンドウは、「動作解析」リボン内の「分析」パネルにある「デバッグ」ボタンを押すと表示されるメニューから「デバッグサブウィンドウ」を

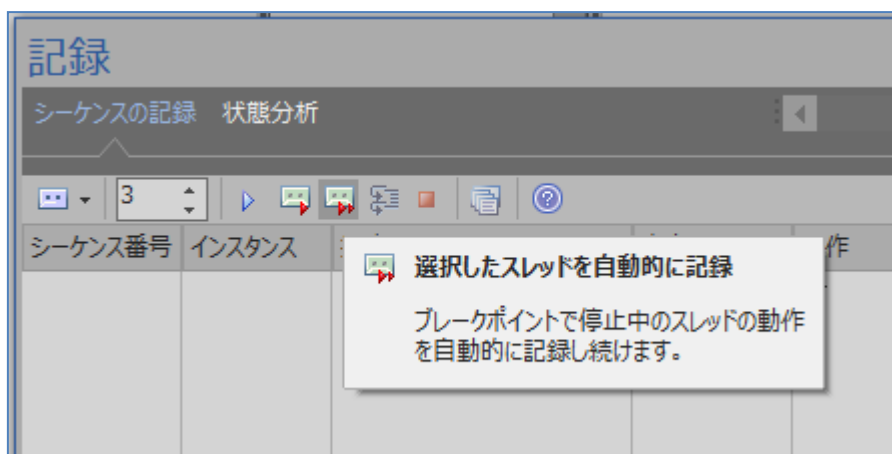
選択すると表示できます。

もし、ブレークポイントを示す赤丸に実行時に「？」マークが表示される場合は、設定が適切に行われていないか、デバッグ情報がないアプリケーションを実行しているため、ブレークポイントの位置が確定できない状況になっています。ほとんどの場合は、デバッグ版ではない(デバッグ情報を含まない)アプリケーションを指定しているか、Enterprise Architect に読み込んだソースのパスと、デバッグ版に含まれるパスの情報が一致していないかのいずれかが原因です。

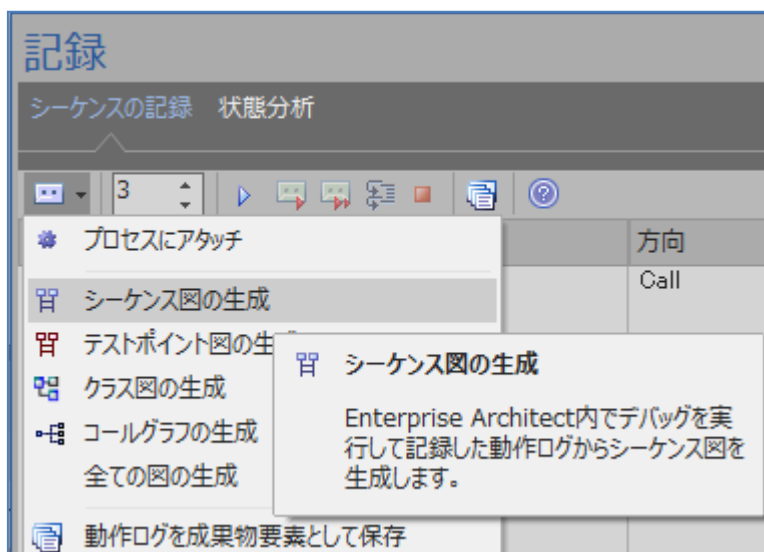
なお、DLL を対象にしている場合には、その DLL が読み込まれるまでは、「？」マークのままになっています。DLL がロードされたタイミングでブレークポイントの位置の同定を行い、位置が確定できれば赤丸で表示されます。

ブレークポイントに停止したら、記録を終了する位置にブレークポイントを追加します。最後まで記録する場合には、ブレークポイントの追加は不要です。

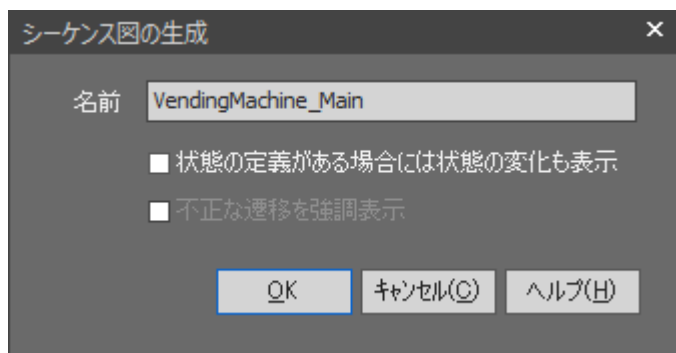
その後、動作履歴の記録を開始します。記録サブウィンドウのツールバーにある「選択したスレッドを自動的に記録」ボタンを押すと、対象のスレッドが終了するか、次のブレークポイントに達するまで自動的に記録が行われます。

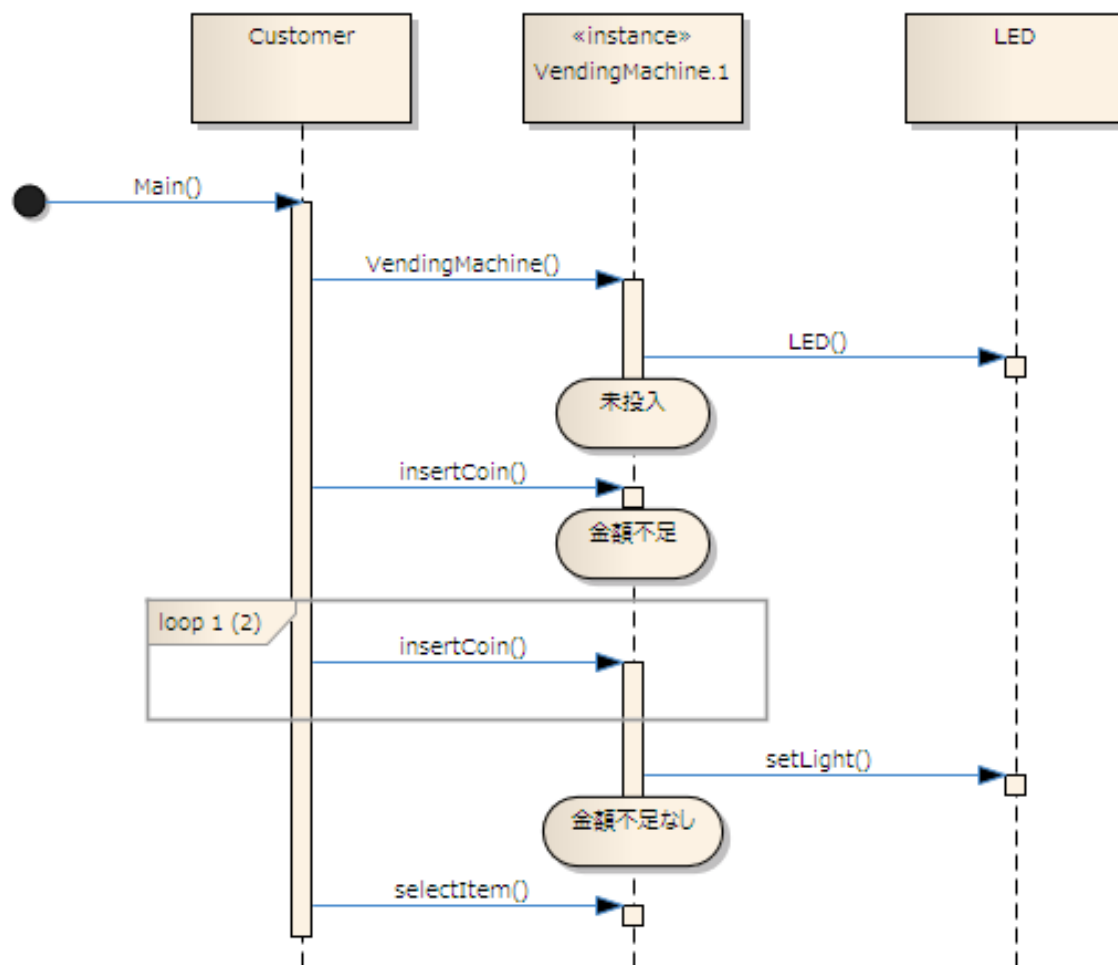


記録が終わったら、デバッガの動作を終了させてください。その後、記録サブウィンドウの一覧内で右クリックして「シーケンス図の生成」を選択するか、「記録」サブウィンドウのツールバーの一番左端のボタンを押すと表示されるメニューの「シーケンス図の生成」を選択してください。



以下のような画面が表示されますので、「OK」を選択するとシーケンス図が生成されます。ここで、状態の変化も記録する設定にしている場合には、状態の情報もシーケンス図に表現されます。このような、状態の記録機能の詳細については、ヘルプファイルをご覧ください。





上の例は、状態の記録を有効にした場合の生成結果です。シーケンス図内に状態要素が配置され、状態の変化がわかりやすく表現されています。なお、ステートマシン図で状態間に遷移がない場合には、状態要素の外枠が赤色で表示され、設計情報と異なる動作をしていることがわかるようになっています。

なお、記録サブウィンドウで右クリックしてコンテキストメニューを表示し、「動作ログをファイルに保存する」を実行することで、動作履歴を XML 形式で保存できます。保存したログファイルは、同じプロジェクトで「動作ログをファイルから読み込む」を実行すれば、再度内容を解析したりシーケンス図の自動生成を実行したりすることができます。

6 マーカーを利用したシーケンス図の作成

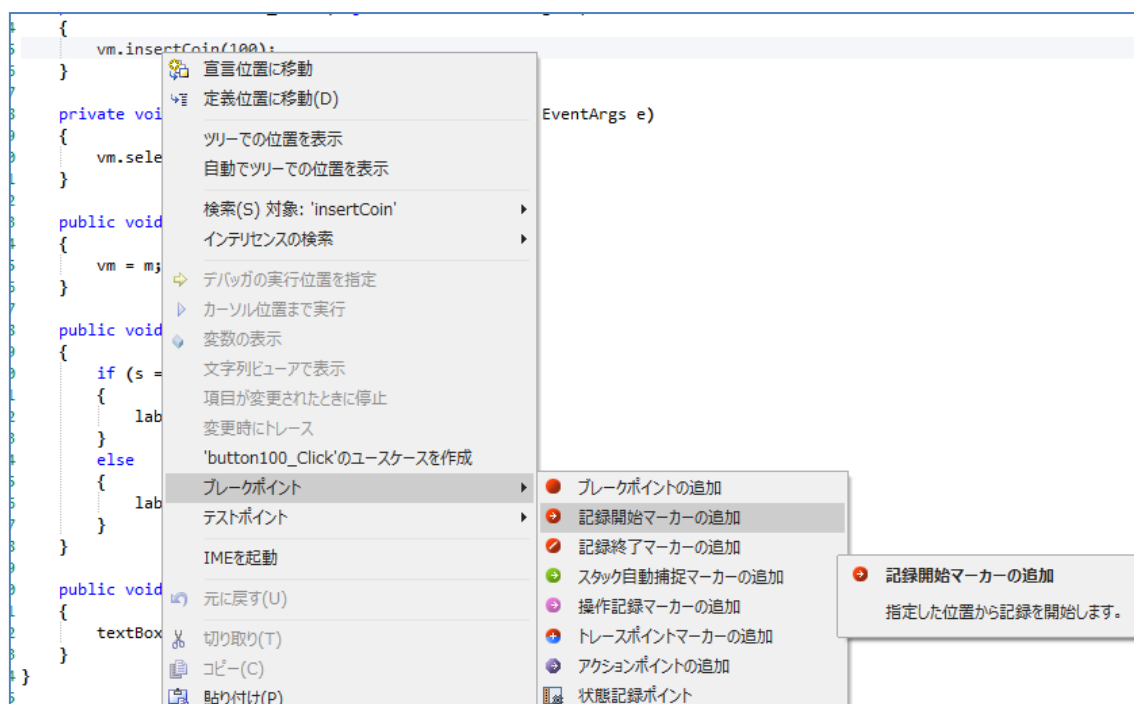
第 5 章では、ソースコードにブレークポイントを設定し、プログラムの実行が停止した時点で、手動でログの記録を行っていました。単純なアプリケーションの場合にはこの方

法で対応できますが、複数のスレッドが同時に実行されていて、その全てについて記録を行いたい場合や、ブレークポイントで停止させると実行に影響するような場合もあります。

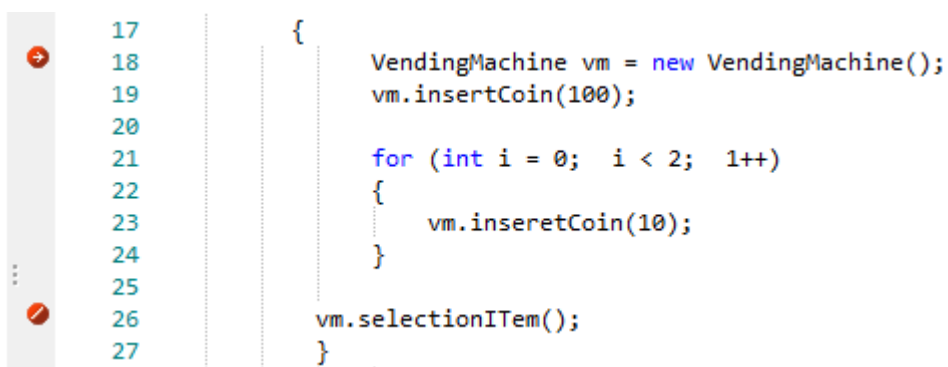
(アプリケーションによっては、Enterprise Architect のデバッガで実行したり、ブレークポイントで停止させたりすると、処理速度やタイミングが変わる関係で通常の動作と変わってしまったり、不正終了してしまったりする場合があります。)

このような場合には、「マーカー」の機能を利用します。マーカーを利用すると、ブレークポイントのように停止することなく、指定した範囲の動作ログを取得することができます。特に、マルチスレッドの処理を記録したい場合には、マーカーの使用が必須となります。(ブレークポイントを利用した場合には、停止したスレッドのみが記録対象になります。)

マーカーを設定するには、Enterprise Architect 内部でソースコードを表示した状態で、マーカーを配置する行で右クリックします。次のようなコンテキストメニューが表示されます。



このメニューの中の「記録開始マーカーの追加」「記録終了マーカーの追加」などを選択すると配置される記号をマーカーと呼びます。例えば、次のように設定した場合には、記録開始と終了の範囲の処理を自動的に記録します。



```
17 {
18     VendingMachine vm = new VendingMachine();
19     vm.insertCoin(100);
20
21     for (int i = 0; i < 2; i++)
22     {
23         vm.inseretCoin(10);
24     }
25
26     vm.selectionITem();
27 }
```

このマーカーは複数設定することができますので、状況に応じて興味のある範囲にマーカーを設置し、動作結果をシーケンス図として表現すると良いでしょう。

○改版履歴

- 2007/07/11 Enterprise Architect バージョン 7.0 リリースに伴い、内容を更新。
- 2008/03/06 Enterprise Architect バージョン 7.1 リリースに伴い、内容を更新。「マーカー」についての説明を追加。
- 2008/08/29 Java のバージョンの誤記を修正。
- 2009/03/24 Enterprise Architect バージョン 7.5 リリースに伴い、内容を更新。
- 2009/06/18 利用条件についての説明を追加。
- 2009/12/03 冒頭の説明文について、内容を追加。その他、いくつかの補足を追加。
- 2010/03/29 「注意点」を追加。
- 2010/04/16 Enterprise Architect バージョン 8.0 のリリースに伴い、内容を更新。
- 2010/06/30 VB・C 言語についての注意書きを追加。
- 2011/02/18 .NET 言語について、言語の種類を明示。
- 2011/05/18 Enterprise Architect バージョン 9.0 のリリースに伴い、内容を更新。
- 2011/12/08 Enterprise Architect バージョン 9.2 のリリースに伴い、内容を更新。
- 2012/03/07 Enterprise Architect バージョン 9.3 のリリースに伴い、内容を更新。
- 2012/12/14 Enterprise Architect バージョン 10.0 のリリースに伴い、内容を更新。
- 2013/01/07 GDB や PHP などの場合の補足を追加するなど内容の微修正。
- 2013/11/15 動作解析の設定方法など、ビルド 1009 までに変更になった点の修正。
- 2013/11/21 動作解析の設定画面の差し替え。説明の微修正。
- 2014/01/17 2 ページ目の説明内容の改善。
- 2014/01/29 説明文を全体的に見直し。
- 2014/02/20 いくつかの補足説明を追記。
- 2015/02/12 Enterprise Architect バージョン 12.0 リリースに伴い、内容を更新。
- 2016/10/07 Enterprise Architect バージョン 13.0 リリースに伴い、内容を更新。
- 2018/05/16 Enterprise Architect バージョン 14.0 リリースに伴い、内容を更新。
- 2018/09/13 Enterprise Architect バージョン 14.1 リリースに伴い、内容を更新。また、ドキュメント「ソースコードの生成と読み込み 機能ガイド」と重複する内容を削除。