ソフトウェア工学 58－16
（１９８８．２．５）

ソフトウェア変更支援システム

山田宏之　手塚慶一
大阪大学　工学部

一部の変更に伴う他の部分への影響（相互作用）の波及を指摘するソフトウェア変更支援システムについて検討する．我々は相互作用に関する情報を直接的に記述できる相互作用記述言語を提案してきた．本システムは，この言語で記述された相互作用記述とソフトウェアの機能情報とをソフトウェアのドキュメント情報として利用する．本システムは４つのモジュールと２つのファイルから構成される．まず，オブジェクト指向型言語で記述された待ち行列シミュレーションにおいて，モデル変更時に相互作用記述に基づいたシステムの指示にしたがって，変更に必要な情報をユーザがシステムに与えるだけでその変更が達成されることを示す．さらに，ソフトウェアの構造に基づいて機能情報からソフトウェアの内容に対する説明を生成する機能について述べる．

A Knowledge-Based System for Software Modification

Hiroyuki YAMADA and Yoshikazu TEZUKA

Faculty of Engineering, Osaka University, 2-1 Yamadaoka, Suita-shi, 565 JAPAN

This paper presents a knowledge-based system for software modification, which can point out interaction (the side effect caused by modifying the software). We have proposed Interaction Representation Language (IRL) that consists of high-level primitives corresponding to concepts relevant to the management of the interaction. This system uses the documentation that consists of Interaction Representation (IR) and Functional Information (FI). Queueing simulation software written in an object oriented language shows IRL is more prospective to make the software modification easier. Furthermore, this paper shows the explanation facility for supporting software understanding. The explanation is generated by functional information and structural information of the software.

1．はじめに

ソフトウェアを継承して再利用する場合に，ソフトウェアの内容（機能および構造）を理解する必要があるが，他人が開発したソフトウェアを把握する手段にはそのドキュメントあるいはソースコードを用いているのが現状であり，多大な労力が必要である．また，ソフトウェアの不用意な変更は他の部分に対する変更を引き起こす可能性があるために，既存のソフトウェアを修正することは容易でない．我々は，このソフトウェアの変更時に生じる他の部分への影響を相互作用と呼び，相互作用の波及を指摘することによりソフトウェアの修正を容易にするために相互作用に関する情報を記述するための言語（相互作用記述言語：Interaction Representation Language，ＩＲＬ）を提案した［1］．そして，この言語で記述された相互作用記述（Interaction Representation：ＩＲ）とソフトウェアの機能情報とをソフトウェアのドキュメント情報として計算機が利用することにより，変更時にユーザを支援するシステムの構成について検討している．そこで，本稿ではシステムの対象としてオブジェクト指向型言語で記述されたフェリーシミュレーションソフトウェアをとりあげ，そのシミュレーションモデルの変更時に相互作用記述に基づいたシステムの指示にしたがって，変更に必要な情報をユーザがシステムに与えるだけでその変更が達成されることを示す．さらに，ソフトウェアの変更時に対象ソフトウェア記述言語については理解しているが対象ソフトウェアの内容を知らないユーザに対して，ソフトウェアの内容を把握させるためにソフトウェアの機能情報と構造情報とを用いたソフトウェア説明生成について述べる．

2．実験システムの概要

2．1　実験システムの構成

本システム構成を図１に示す．本システムは４つのモジュール（相互作用検出モジュール，対話インタフェースモジュール，修正モジュール，説明生成モジュール）と２つのファイル（ドキュメンテーションファイル，対象ソフトウェアファイル）から構成される．

各モジュールの機能とファイルの内容について説明する．

（１）相互作用検出モジュール
（Interaction Detector）

このモジュールは本システムの中心部であり，システム全体の流れを制御する．変更に関連するＩＲはドキュメンテーションファイル内のドキュメンテーションクラスのクラス階層に基づいて探索される．

まず，変更に直接関係するクラスのドキュメンテーションクラスが探索される．もし，有効な情報が無ければ，その上位クラスのドキュメンテーションクラスからＩＲが探索される．探索されたＩＲは条件スロットの内容がテストされ，真のもののみが有効となる．

（２）対話インタフェースモジュール
（Dialog Interface）

このモジュールは，システムがソフトウェアの変更要求をユーザから受けるときあるいは相互作用検出モジュールがＩＲを解釈するときに，必要に応じてシステムとユーザとの間で質問・応答を行う．現段階では，システムが出す指示はあらかじめ自然言語による文で準備しており，ユーザからの入力は選択枝あるいは名詞に限られる．

（３）修正モジュール（Editor）

このモジュールは，相互作用検出モジュールからの情報をもとに，対象ソフトウェアにおける相互作用の波及部分を対象ソフトウェアファイルから検索し，該当部をエディタ上に読み込み，ユーザに修正の場を提供する．変更終了後，変更情報を相互作用検出モジュールに渡す．

（４）説明生成モジュール
（Explanation Generator）

このモジュールは相互作用の波及が指摘されたとき，その波及の理由あるいは波及箇所の変更時に必要な情報をユーザに説明する．

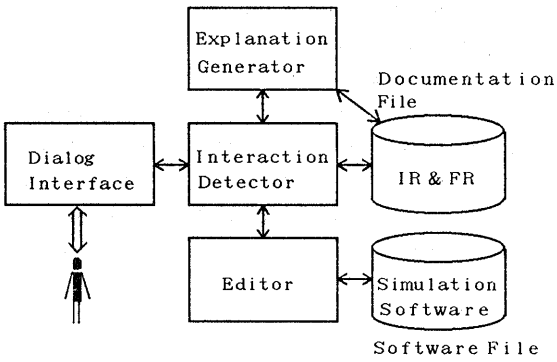

図１　システム構成図

また，このモジュールは，ユーザの要求に応じてメソッドの機能記述とメソッドの構造情報とからソフトウェア内容についての説明を生成する．

（5）対象ソフトウェア・ファイル
(Software File)

このファイルは，対象ソフトウェアのソースコードを持つ．本対象ソフトウェアは文献［2］を参考にわれわれが開発したオブジェクト指向型言語に基づいて記述されている．その詳細は文献［3］を参照されたい．

（6）ドキュメンテーション・ファイル
(Documentation File)

このファイルは，クラスに関する情報が記述されるドキュメンテーションクラスをもつ．

ドキュメンテーションクラスは，図2に示すように各クラスに一つずつあり，そのクラスに関する管理情報をもつために，メタクラスに相当する．

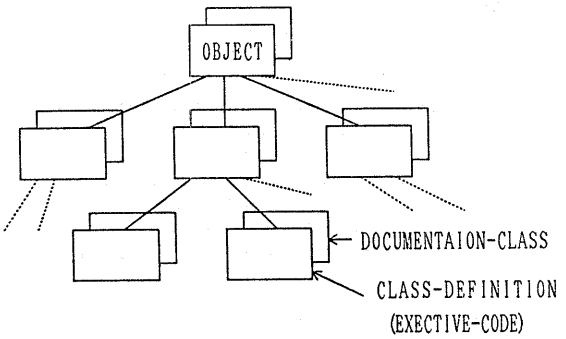

図2　クラスとドキュメンテーションクラス

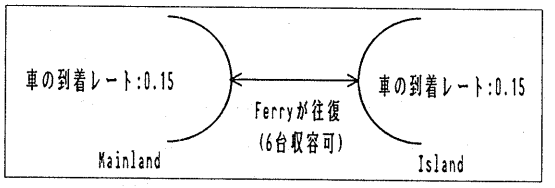

図3　フェリーシミュレーションのモデル

2．2　システムのファイル構成

（1）対象ソフトウェアファイル

本実験システムの変更支援の対象となるシミュレーションモデルの例としては，図3に示すようにあるフェリーが本島と島との間を往復して車を運ぶフェリーシミュレーションを考える．このシミュレーションモデルを以下に簡単に説明する．

フェリーのサービスは午前7時から始まり，午後7時に本島側で終了する．フェリーは6台の車を収容することができる．フェリーは，車を6台積むとき，あるいは，待ち行列に車が無くなったときに対岸に向けて出航する．本島および島への車の到着間隔は双方とも平均0．15の指数分布とする．また，フェリーの渡航時間は平均8，偏差0．5の一様分布とする．

本シミュレーションソフトウェアのクラス階層を図4に示す．この図において，実際のシミュレーションモデルが記述されるクラスはクラスIslandArrival，クラスMainlandArrivalそしてクラスFerrySimulationであり，それ以外のクラスには，前述のクラス定義をするときに用いるメソッド，シミュレーションを実行管理するために用いるメソッド等が定義される．クラスIslandArrivalには島に到着する車のタスクが定義され，クラスFerryには本島と島とを往復するフェリーのタスクが定義される．クラス FerrySimulationには個々のシミュレーションオブジェクトの到着間隔とシミュレーションに利用されるリソースが定義される．

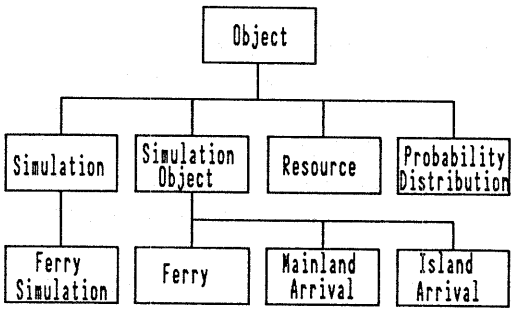

図4　フェリーシミュレーションのクラス階層

（2）ドキュメンテーションファイル

ドキュメンテーションクラスの記述形式を図5に示す．ドキュメンテーションクラスには，クラスに関する情報（上位クラス名，クラス変数名，インスタンス変数名），そのクラスがもつメソッドに関する情報（＜メソッド記述＞）およびＩＲから構成される．＜メソッド記述＞は，メッセージパターン，仮引数，機能記述（ＦＲ）から構成される．機能記述は，そのメソッドの働きについての説明が記述される．

```
<meta-class>::=
    (<class name> meta-class
      (superclass (<class name>))
      (class_variable (<variable name>)*)
      (instance_variable (<variable name>)*)
      (class_method_reference
            <method description>*)
      (instance_method_reference
            <method description>*)
      (interaction_representation
            (<interaction representation>)) )

<method description>::=
    ( (message_pattern (<message pattern>))
      (formal_argument (<argument_name>)*)
      (functional_information
            (<functional description>)) ) )
```

図5　ドキュメンテーションクラスの記述形式

ＩＲは，相互作用に関連する概念が直接的に記述されたもので，３つの関係記述（クラス関係，メソッド関係，キーワード関係）で構成される．換言すれば，ＩＲはオブジェクト間の機能的な結び付きが記述されたものである．相互作用記述言語の詳細は文献［１］を参照されたい．

２．３　実験システムの処理手順

本システムは，次の手順によりソフトウェアを変更する．

（１）ユーザが対話インタフェースモジュールにより変更要求および変更箇所をシステムに伝える．現システムはメニュー選択形式により，行われる．

（２）システムは修正モジュールを起動させて変更箇所のコードをエディタ上に読み込み，ユーザに示す．新たにクラスを定義するときは，クラス定義のスケルトンを示す．

（３）ユーザがその箇所を変更すると相互作用検出モジュールがその変更に関連するＩＲをドキュメンテーションファイルから探索する．

（４）ＩＲが存在するとき，相互作用検出モジュールにより相互作用の影響が及ぶ箇所を検出し，ユーザに指摘し，以下の作業を実行する．影響が及ぶ箇所が複数存在する場合には，個々に以下の作業を実行し，すべての箇所が修正されたとき，現在扱っている変更に伴う相互作用に対する処理を終了する．

（４－１）ユーザが指摘箇所を変更する必要がないとき（４）に戻る．

（４－２）ユーザが指摘箇所を変更するとき，修正モジュールが起動され，変更箇所を対象ソフトウェアファイルから検索し，ユーザに示す．

（４－３）ユーザがその箇所を変更すると相互作用検出モジュールがその変更に関連するＩＲをドキュメンテーションファイルから探索し，ＩＲにしたがって相互作用の影響が及ぶ箇所をユーザに示す．もし，有効なＩＲが存在せず，ユーザからも相互作用の影響に関する情報がなければ現在対象としている変更に関する相互作用の処理が終了し（４）に戻る．そうでなければ，（４－３）で行った変更に伴う影響が及ぶ箇所に対して（４）からの作業を再帰的に実行する．

（５）ＩＲが存在しないときシステムが指摘できなかった相互作用に関する情報をユーザから指摘されなければ処理を終了する．そうでなければ（４）へ戻る．

３．待ち行列シミュレーションソフトウェアの変更支援

３．１　待ち行列シミュレーションモデルの変更例

シミュレーションモデルの変更例として，図６に示すようにこのモデルに別な島に到着する車を追加し，フェリーのタスクを本島と２つの島の間を循環して隣の島へ車を運ぶように変更することを考える．このとき新しい島へ到着する車が実行するタスクは既存の島へ到着する車と同じであり，他の条件の変更はないものとする．

変更後のクラス階層を図７に示す．このとき，ユーザは既存のソフトウェアに対して，新しい島に到着する車を表すクラスを定義し，クラス Ferryのタスクを変更しなければならない．

３．２　ソフトウェア変更支援

以下，ユーザが新しい島に到着する車を表すクラスIslandArrival2を付加したときのシステムの支援例について述べる．実行例を図８に示す．ここで下線部分はユーザによる入力である．

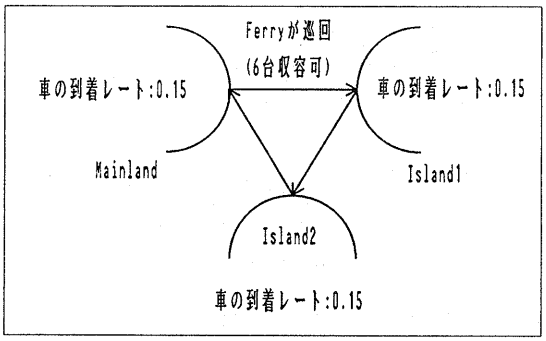

図６　フェリーシミュレーションのモデル変更例

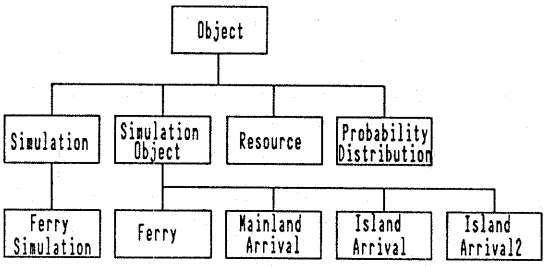

図７　変更後のクラス階層

```
$(scss)

何をしますか？
追加（A）；変更（C）；削除（D）；終了（Q）？ A
何の追加をしますか？
クラス（C）；メソッド（M）；リソース（R）；キーワード（K）；その他（O）？ C

追加対象となるCLASSの名前を教えて下さい．
IslandArrival2

新しいクラス　IslandArrival2を定義します．
よろしいですか？（Y/N)Y

                    ┆（クラスの定義）

クラス IslandArrival2が新たに定義されました．
次の条件が成立する場合はT，しない場合はFと答えて下さい．

追加されたオブジェクトはシミュレーションに到着する? t

CLASS IslandArrival2の追加により，クラス FerrySimulationのメソッド defineArrival
Scheduleに相互作用が発生する可能性があります．

どうしますか？
修正（M）；理由（R）；提示（S）；情報（I）？ m

                    ┆（メソッドの修正）

次の条件が成立する場合はT，しない場合はFと答えて下さい．

メッセージパターンが変更された? f

クラス FerrySimulationのメソッド defineArrivalScheduleの変更に伴う相互作用は他に
ありませんか？（Y/N）N
他の変更がありますか？（Y/N）Y
```

図８　システムの実行例

まず，システムに変更箇所として新しいクラスを追加することを伝え，修正モジュールを起動させ，ユーザが新しいクラスのコードを入力する．

相互作用検出モジュールはクラスの付加に伴う相互作用の波及を検出するために，このクラスが付加されるクラスSimulationObjectのドキュメンテーションクラス内に存在するＩＲを探索する．ここでは，クラスの付加より，クラス関係の'addition'に記述されている情報が引き出される（図９参照）．システムは，この情報内の条件スロットの内容をユーザに示し，この内容の真偽を問い合わせる．ユーザは，この質問に対して真（T）か偽（F）か答える．ここでは，ユーザがＴと答えているので，クラスFerrySimulationのメソッド 'defineArrivalSchedule'にクラスの追加に伴う相互作用波及の可能性が指摘されている．このメソッドを修正するときには，修正モジュールが起動され，このメソッドをエディタ上に読み込み，ユーザ自身が修正する．

このメソッドが修正されたとき，メソッドの変更に伴う相互作用波及の有無を調べるために，修正されたメソッドを持つクラス FerrySimulationに対するドキュメンテーションクラスからＩＲが探索され，メソッド関係が参照される（図１０参照）．この情報は，相互作用がクラス Simulation内のメソッド'startUp'に波及する可能性を示すが，条件スロットのテストより，この情報は無効となる．そして，システム内に有効な別のＩＲが存在しないために，システムはユーザにメソッドの変更に伴う相互作用の波及の有無を問い合わせ，なければ追加に伴う相互作用に対する処理が終了する．

リソースの追加に伴う相互作用に対しても同様に処理される．結局，ユーザは，システムの指示に従って，３箇所を修正することにより，所望のソフトウェアに変更することができる．

```
(class_relation
   (addition
      ((condition
          (追加されたオブジェクトはシミュレーションに到着する))
       (operation (defineArrivalSchedule (FerrySimulation)))
       (reason (シミュレーションへの到着が定義される))
       (required_information
          (time (到着時刻))
          (probability_distribution (到着間隔の分布)) ) ) )
   (deletion
      ((condition
          (削除されるオブジェクトはシミュレーションに到着していた))
       (operation (defineArrivalSchedule (FerrySimulation)))
       (reason (シミュレーションへの到着が定義されている))
       (required_information ()) ) ) )
```

図９　クラスSimulationObjectに関する相互作用記述一部

```
(method_relation
   (defineArrivalSchedule
     ((condition (メッセージパターンが変更された))
      (operation (startUp (Simulation)))
      (reason (このメソッド内で変更されたメソッドを参照している))
      (required_information
         (pattern (変更後のメッセージパターン)))) )

   (defineResources
     ((condition (メッセージパターンが変更された))
      (operation (startUp (Simulation)))
      (reason (このメソッド内で変更されたメソッドを参照している))
      (required_information
         (pattern (変更後のメッセージパターン)))) ) )
```

図１０　クラスFerrySimulationに関する相互作用記述一部

３．３　ソフトウェアの説明生成

本システムは相互作用が指摘されたとき，システムに以下の項目に対する問い合わせができる．

（１）変更時に用いられる情報

修正するときにどのような情報が必要で，それはどのように利用されるか？

（２）相互作用の波及理由

なぜ相互作用が波及したか？

（３）メソッドの内容

メソッドがどんな機能をもち，どのように構成されているか？

（１）の質問に対しては，システムはＩＲ内の情報スロット内の説明部を用いて答えを返す．さらに，必要な情報とその情報を利用するメソッドとの組合せが記述されているので，必要に応じて説明生成モジュールにより情報を利用するメソッドの説明を生成する．

（２）の質問に対しては，システムは理由スロットを用いて理由を示す．

（３）の質問に対しては，直接そのコードを見せる代わりに，説明生成モジュールによりメソッドの機能記述（Functional Representation）および構造情報（Structural Information）とからメソッドの内容について説明を生成する．

説明生成の手順

メソッドがトップダウン的に記述されている場合，ある機能を持つメソッドは，より簡単な機能を持つメソッドの組合せにより構成される．そこで，本システムでは，メソッドの構造に基づいてその説明をトップダウン的に生成する．その概念を図１１に示す．

以下に，説明生成における概念的な手順を述べる．

（１）指定されたメソッドの機能記述をドキュメン

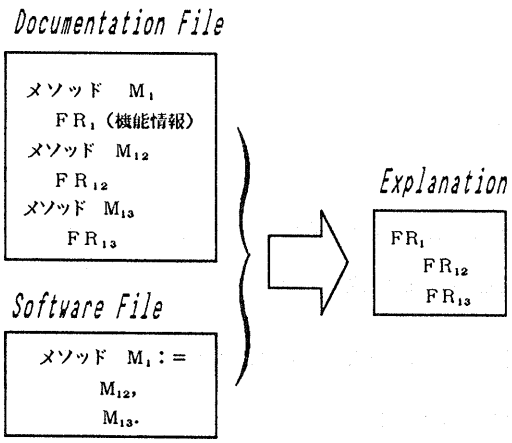

図１１　説明生成の概念図

テーションクラスから取り出し，そのメソッドの説明としてユーザに提示する．

（２）ユーザに対して，メソッドの説明レベルが不十分な場合には，メソッド構造に基づいて，メソッドを下位メソッドに分解する．

（３）分解された各下位メソッドの機能記述から，そのメソッドの構造情報を基に説明を生成する．

（４）もし，説明の中にユーザにとって説明のレベルが不十分なメソッドがあれば，（２）へもどる．そうでなければ，終了する．

説明の具体例を図１２に示す．ここで，図１２（b）では，メソッド “defineArrivalSchedule” に対する機能情報のみによる説明が示されている．ユーザがこのメソッドの構造を知りたい場合には，(c)，(d)の機能情報を用いて(e)のソースコードの構造をもとに，その説明(f)が生成される．

本説明生成において，メソッドは対象記述言語のプリミティブメソッド，もしくは，ユーザが理解できるメソッドが現れるまで分解される．したがって，トップダウン的な説明生成により，説明の詳細さをユーザのレベルに合わせて自由に設定できると考えられる．

また，メソッドの機能記述はそのメソッドの振舞いを短文，あるいは，キーワードで表現したものであり，ソフトウェア開発者により記述される．説明時にメソッドのソースコードをユーザに示す代わりに，人間により直接記述されたメソッドの機能記述を用いることにより，説明の概念レベルを向上させ，ユーザにとって理解し易い説明ができると考えられる．

４．むすび

本稿ではソフトウェアの変更時に，システムとユーザとの間でフレンドリーなインタラクションが実現できるように定義された相互作用記述を用いて，待ち行列シミュレーションソフトウェアを対象としてそのインタラクションを実現するソフトウェア変更支援システムについて述べた．本システムでは，ソフトウェアを変更すると，ＩＲＬにより記述された相互作用に関する情報に基づいて，ソフトウェアの変更に必要な情報だけをユーザから得るだけで，その変更操作が達成される．

また，本システムではソフトウェアの変更時にユーザがソフトウェアの内容を把握することを支援するために，ソフトウェアの機能情報によりソフトウェアの構造に基づいてその説明を生成する機構を述べた．

今後の課題としては，ソフトウェアの把握をより支援するために，さまざまなアプリケーションに対応したＩＲを記述する上での支援機構ならびにユーザのレベルに応じたより柔軟な説明機能の実現等がある．

［謝　　辞］

本研究を進めるにあたり，有益な御助言を頂いた本学産業科学研究所角所 収教授，山口高平助手に深く感謝致します．

［参考文献］

[1] 山田，山口，真田，角所，手塚：” ソフトウェアの変更におけるユーザインタフェースの高度化を目指す記述言語”，信学論（Ｄ），J70-D, No.11, pp. 2308-2313 (1987).

[2] Goldberg, A. and Robinson, D. : ”Smalltalk-80 The Language and Its Implementation”, Addison Weslay (1983).

[3] Yamada, H. et al : ”Knowledge Based Software Develpment Support System for Queueing Network Simulation”, Proc. of PCCS, KAIST, pp. 348-350, Seoul (1985).

[4] 山田，手塚：” ドキュメンテーションを利用したソフトウェア説明機能”，情報処理学会 第３５回全国大会，5N-1 (1987).

[5] Swartout, B. : ”GIST English Generator”, AAAI-83, pp.404-409 (1983).

```
((message_pattern (defineArrivalSchedule))
 (formal_argument ())
 (internal_variable ())
 (functional_information
   (シミュレーションオブジェクトが確率分布関数に基づいて規定の時間間隔でシミュレーションにはいることを定義する)))
```

(a)メソッドdefineArrivalScheduleの情報の一部

メソッドdefineArrivalScheduleは
1 (シミュレーションオブジェクトが確率分布関数に基づいて規定の時間間隔でシミュレーションにはいることを定義する)

(b)メソッドの機能情報による説明

```
(message_pattern ((scheduleArrivalOf_accordingTo_startingAt))
(formal_argument ((aSimulationObjectClass)
                  (aProbabilityDistribution)
                  (timeInteger))
(internal_argument ())
(functional_information
  (^aSimulationObjectClass のインスタンスが ^aProbabilityDistribution に基づいて指定の時間間隔で, 指定された時刻 ^timeInteger からシミュレーションに入るように定義する))
)
```

(c)メソッドscheduleArrivalOf_accorfingTo_startingAtの情報の一部

```
((message_pattern ((scheduleArrivalOf_at))
 (formal_argument ((aSimulationObject)
                   (timeInteger))
 (internal_argument ())
 (functional_information
   (^aSimulationObject が指定された時刻 ^timeInteger にシミュレーションに入るように定義する)))
```

(d)メソッドscheduleArrivalOf_atの情報の一部

```
(defineArrivalSchedule ()
        (send self scheduleArrivalOf_accordingTo_startingAt
                MainlandArrival (send Exponential parameter 0.15)
                420.0)
        (send self scheduleArrivalOf_accordingTo_startingAt
                IslandArrival (send Exponential parameter 0.15)
                420.0)
        (send self scheduleArrivalOf_at
                (send *Ferry new) 420.0))
```

(e)メソッドdefineArrivalScheduleの構造

1 のメソッドは,
1-1 (MainlandArrival のインスタンスが (平均 0.15 の Exponential 分布) に基づいて指定の時間間隔で, 指定された時刻 420.0 からシミュレーションに入るように定義する)
1-2 (IslandArrival のインスタンスが (平均 0.15 の Exponential 分布) に基づいて指定の時間間隔で, 指定された時刻 420.0 からシミュレーションに入るように定義する)
1-3 ((Ferry のインスタンス) が指定された時刻 420.0 にシミュレーションに入るように定義する)

(f)メソッドの構造に基づいた説明

図１２　メソッドdefineArrivalScheduleの説明生成