目录

面向 BPEL 服务组装程序的变异测试系统 (简称 μBPEL) V1.0 文档

1.背景介绍

面向服务的架构(Services-Oriented Architecture, SOA)的核心是将系统的每一个功能模块进行封装并定义为服务，可以通过接口调用该服务。Web 服务是 SOA 基于 Web 的一种实现，Web 服务是描述一系列操作的接口，使用标准的、规范的 XML 描述接口，这一描述中包括与服务进行交互所需要的全部细节，而对外的接口中隐藏了服务实现的细节，仅仅提供一系列可执行的操作。BPEL(Business Process Execution Language)是一种可执行的服务组装语言，可以将多个相互交互的 Web 服务编制在一起形成业务流程，并作为一种更为复杂的 Web 服务对外提供服务。由于 Web 服务相互协作的动态性，互联网环境的松耦合特性、开放的运行环境，如何保证服务组装的正确性与可靠性尤显重要。

软件测试是一种保证软件可靠性的重要手段。测试用例质量决定着软件测试能否能高效完成，变异测试是检验测试用例质量的有效手段。BPEL 程序的手工测试非常困难，因此有必要研究面向 BPEL 程序的自动化测试技术。

1.1 BPEL

BPEL 是一种使用 Web 服务定义和执行业务流程的可执行编程语言。用于实现服务组合以提供复杂 Web 服务。BPEL 程序基本组成包括变量、伙伴链接、活动等。具体介绍如下：

(1) **变量**：主要用来存储消息和对消息进行重新格式化和装换的。它作为参数被服务调用或者接受来返回结果值。

(2) **合作伙伴链接**：用于实现与 Web 服务的交互。其中伙伴指那些与 BPEL 流程交互的服务，而伙伴链接内容包括两个合作伙伴通信交换信息。合作伙伴有 myRole 和 partnerRole 这两种属性，分别指流程服务和伙伴服务提供者的角色。

(3) **活动**：BPEL 业务流程的基本构成单位。BPEL 语言的基本单元为活动，活动按作用可以分为：基本活动、结构活动、特殊活动和故障处理。其中基本活动是指与外界进行交互的最简单的形式。它主要实现流程的基本功能。基本活动主要包括：用于与外界进行交互的基本活动，如 invoke、receive、reply；用于传输数据的活动 assign；用于发出故障信号活的 throw；

用于等待一段时间或到达某一时间再继续的活动 wait；不执行任何操作的活动 empty；终端所有流程实例的活动 terminate 等。结构活动则是规定一组活动发生的顺序。主要是控制流程的结构，描述业务流程是如何通过把它执行的基本活动组成结构而被创建的，这些结构表达了设计业务协议的流程实例间的控制形式、数据流程、故障和外部事件的处理以及消息交换的协调。主要包括：活动间的顺序控制，如 sequence、switch 和 while；活动间的并发和同步 flow 结构；基于外部事件的不确定的选择由 pick 来提供等。

(4) **故障处理机制**：任何流程在执行过程中都可能有异常和错误情况发生。BPEL 提供可定制的错误处理和补偿机制，可在定义流程的同时，根据流程自身的特点、异常类型以及实际需求，定义相应的错误恢复或补偿操作，以应对相应的异常情况。BPEL 提供 faultHandlers、catch 和 catchAll 元素作为异常管理机制，类似于 try-catch 结构。故障接受 WSDL 定义的故障消息来生成，或者通过使用 throw 元素触发。并由 catchAll 提供错误处理活动。对于流程中某个事件或错误发生而导致流程取消，已经完成的活动需要复原，这是需要补偿机制，BPEL 提供 Compensation 用来还原已经完成的活动。

由此可见，BPEL 程序主体主要包含两种类型的操作，一个是用于调用和从 Web 服务接收调用的原始程序和数据操作，另一个是用于按计划控制在一个 BPEL 程序中执行那些操作步骤的结构化操作。与传统的应用程序相比，BPEL 程序有其如下特点：

(1) 提供一种显式的集成机制组装 Web 服务。

(2) 参与组装的服务可以采用不同语言实现。

(3) BPEL 程序表示为 XML 文件，不同于传统应用程序。

(4) 支持并发与同步机制。

1.2 面向 BPEL 服务组装程序的变异测试技术

变异测试是一种基于故障的软件测试技术，通过将错误植入程序中来对程序进行测试，具有较强的错误检测能力。广泛用于评估和改进测试用例集完备性和测试技术的有效性。变异测试的基本思想：将变异算子应用于待测程序从而植入错误，得到错误版本的程序集合，这种错误版本的程序被称为变异体。变异算子是对程序设计语言中典型错误的抽象和概括。每个变异体中包含了对原始程序进行的合乎语法的微小改动，从而模拟开发人员编程时所犯的错误。如果一个测试

用例使得变异体的执行行为与待测程序不同，这个变异体就称为“被杀死”，即检测出这种故障。

面向 BPEL 程序的变异测试，首先需要分析 BPEL 语言的特点，找到可能在编写中出现的错误，这些就是所谓的变异算子。由 Estero-Botaro 等人所提出的面向 BPEL 的 26 种变异算子，就是模拟程序员用工具书写 BPEL 可能出现的错误，其是用另一种语法结构替换当前的语法结构，并且保证替换后程序的语法是正确的，但并不保持语义的一致。基于以上变异算子，我们提出一种面向 BPEL 程序的变异测试框架。其通过对 BPEL 文件（.BPEL）进行解析后，将变异算子进行匹配并生成列表，找到对应的变异算子的处理方法，结合 XML 文件读写生成相应的变异体。通过输入的测试用例对原始 BPEL 程序和变异体分别进行测试，统计测试结果。

对于测试结果，通常采用变异得分和故障检测率两个指标来衡量测试用例集合的充分性与有效性。第一个度量指标为变异得分(Mutation Score，*MS*)，其定义为：

$$MS(P,T)=\frac{D}{(M-E)}$$

其中，*P* 为被测程序，*T* 为测试数据集合，*M* 为变异体的数量，*D* 为杀死的变异体数量，*E* 为等价的变异体数量。变异得分定义了测试用例集合 T 能够杀死非等价变异体的比例，度量了测试用例集合的充分性，其得分越高测试的越有效。等价变异体是一类与源程序语义相同且语法不同的且没有任何测试用例能将其杀死的变异体。对于等价变异体，其无助于提高变异分数，一旦识别为等价变异体剩余测试用例就不必再对其进行测试了，从而可以避免不必要的计算资源的浪费。

第二个度量指标是故障检测率(Fault Discovery Rate, FDR),其定义为：

$$FDR(m,T)=\frac{N_f}{N_t-N_i}$$

其中，*m* 为变异体，*T* 为测试用例集合，N_f 为集合 *T* 中能够杀死 *m* 变异体的数量，N_t 为测试用例集合 *T* 的总数量，N_i 为非法测试用例数量。故障检测率定义了测试用例集合 *T* 能够杀死 *m* 变异体的比例，其检测率越高说明测试用例 *T* 杀死 *m* 变异体的几率越大。

1.3 面向场景的测试技术

面向场景的 BPEL 测试技术，实现较高程度的测试用例自动化生成并解决

BPEL 语言自身的并发多分支等测试自动化难点。这里的测试用例包括测试路径和测试数据两个部分。该技术分如下两个步骤完成测试用例的自动生成工作。

（1）基于变换的 BPEL 测试路径生成方法

首先，基于变换的测试路径自动生成技术将 BPEL 按照程序转换为只包含活动与边的抽象图模型 BGM (BPEL Graph Model)，此部分去除 BPEL 中一些无关紧要的信息。然后，基于一系列的变换规则将图模型变为扩展的与或树(Extended AND-OR Tree)结构。由于 BPEL 程序的测试路径是指该程序执行时的一系列基本活动构成的序列，所以只需遍历上述的与或树，便可得到测试路径集。

（2）基于约束求解策略的测试数据自动生成

在上面得到测试路径后，需要分析对应路径的所有约束条件表达式，通过求解然后得到测试用例。约束求解包括如下两个步骤，一是提取约束条件，即从被测试程序中提取出对应路径的所有约束条件表达式；二是求解约束，即采用正确的算法和策略，求出约束条件表达式的若干可行解。这利用开源 Z3 工具自动生成满足线性条件表达式的可行解的方法，该工具可以严格根据约束表达式自动生成满足条件的若干可行解。根据产生的测试路径集合，再针对每条测试路径生成了测试数据，这些输入数据覆盖了测试路径集的所有测试路径，也就得到了测试用例集合。

通过以上技术，可以得到自动生成满足一定覆盖准则的测试用例集。

2.面向 BPEL 服务组装程序的变异测试系统（μBPEL）分析与设计

2.1 需求分析

根据变异测试技术与 BPEL 程序的特点，提出一个 BPEL 执行变异测试框架，如图 1。BPEL 通过编排 Web 服务和其他组合服务来组合服务。如 Web 服务一样，BPEL 使用 WSDL 规范来描述服务。通过接口，可以得到输入和输出变量信息。框架基本组成如图 1 所示。

面向 BPEL 程序的变异测试框架主要包括如下部分：

（1） 测试用例生成

1) 解析 BPEL 程序的服务描述文件识别出服务提供的操作。

2) 解析待测操作的输入类型和约束条件。

3) 根据测试技术或覆盖准则生成测试用例集。

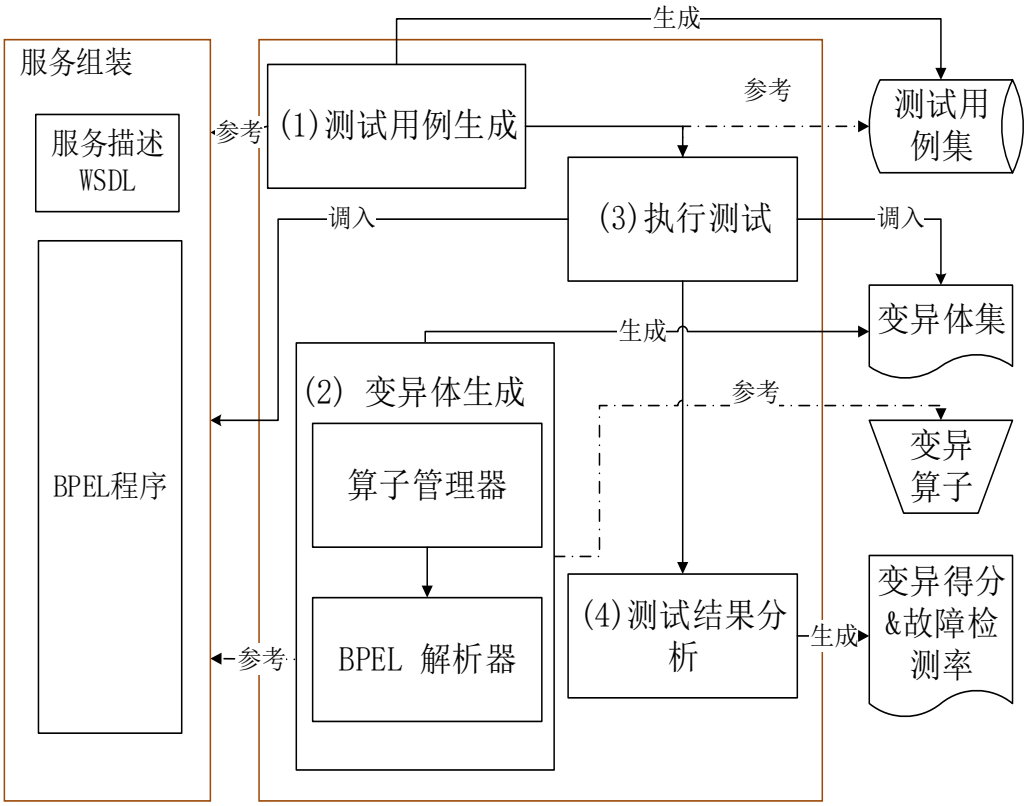

图 1 BPEL 执行变异测试框架

（2） 变异体生成

1) BPEL 解析器对 BPEL 程序进行解析，参考变异算子集，识别出可进行变异的元素及位置。

2) 基于上述解析结果，匹配可应用的变异算子。

3) 根据变异算子的操作规则，运用变异算子管理器，构建相应的变异体。

（3） 执行测试

1) 调入 BPEL 程序，变异体集及生成的测试用例集。

2) 输入测试用例，执行 BPEL 程序并记录结果。

3) 对变异体集中的变异体依次执行相同的测试用例并记录结果。

（4） 测试结果分析

1) 统计结果，计算得出相应的变异得分和故障检测率。

2) 总结、分析和报告结果。

根据上述功能需求的分析，绘制了 μBPEL 的用例图，如图 2 所示。其中，各用例描述如下：

（1） **变异体自动生成**：用户提供待测试 BPEL 程序，系统读取并解析 BPEL 原

始程序进行解析，找到变异点及其相关转换规则，生成变异体。变异体自动生成包括 BPEL 程序解析、变异算子管理、XML 文件读写和变异体生成四个子用例。

1) **BPEL 程序解析：**BPEL 变异体的生成需要在 BPEL 源程序中植入错误，则要对 BPEL 程序解析，鉴别可以变异的元素或指令，记录与变异算子匹配的源代码。BPEL 是在 XML 基础之上建立的语言，在解析的时候利用 Dom4j 解析包，将 BPEL 程序源文件以 Document 返回，并对其进行变异点匹配操作。用户根据输入的 BPEL 程序进行解析，提取出 BPEL 程序节点。
2) **变异算子管理：**系统存储各种转换规则，它通过 BPEL 解析之后得到的变异点，找到相关的植入错误操作，是变异体生成的关键。该用例需要对 34 个变异算子进行有效的管理。
3) **XML 文件读写：**该用例完成对 XML 文件的读写功能，借助于 JAVA 对 XML 的操作来完成 BPEL 的读入解析和错误植入，读取 BPEL 程序利用 Dom4j 解析返回 Document，规则植入时将 Document 变换写入 XML 中，完成 BPEL 的变异体生成。
4) **变异体生成：**该用例通过解析得到的与变异算子匹配的源程序列表，利用算子管理用例中的转换规则，对 BPEL 源程序进行相关的转换，将转换后的信息写入 BEPL 中，该用例同样需要 Dom4j 技术完成 XML 写入操作。

（2） **测试路径生成：**用户提供待测试 BPEL 程序，系统根据该 BPEL 程序生成测试路径。生成测试路径包含解析 BPEL，生成图结构和扩展二叉树和测试路径四个用例。

1) **BPEL 程序解析：**用户根据输入的 BPEL 程序进行解析，提取出 BPEL 程序节点。
2) **转换为图模型：**用户根据提取的信息，将 BPEL 程序转换为只包含顶点与边的图结构。
3) **扩展二叉树：**将图结构进行变换，变换为扩展的二叉树结构，将节点信息保存在文本文件中。
4) **测试路径：**采用一定的遍历算法对二叉树进行遍历，生成满足某种覆盖准则的测试路径。主要包括两种覆盖准则：

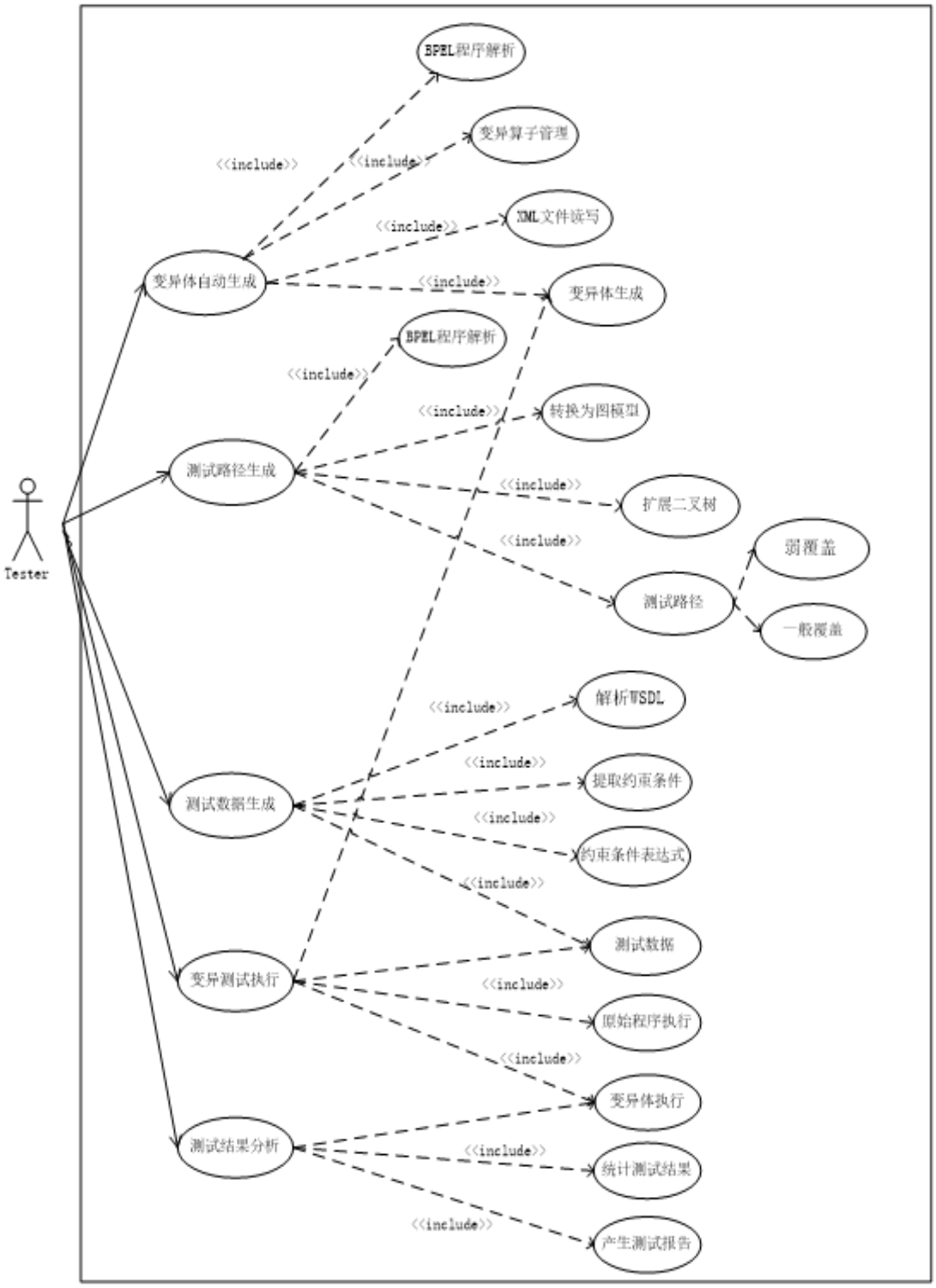

图 2 μBPEL 的用例图

- 弱覆盖准则：该准则在一对并发节点和汇聚节点之间只包括一个可行的并发序列，不考虑并行活动之间的交错处理。

- 一般覆盖准则：该准则在一对并发节点和汇聚节点之间覆盖所有可能的并发序列，不考虑并行活动之间的交错处理。

（3） **测试数据生成：**生成测试路径之后，需要针对每条测试路径生成对应的测试数据。包括解析 WSDL，提取约束条件，约束条件表达式求解和测试数据四个用例。

1)**解析 WSDL：**用户输入待测 Web 服务的 WSDL URI，系统将会自动解析该 WSDL 的信息，分析接口输入与输出的变量及其格式。

2)**提取约束条件：**测试路径生成以后，需要针对每条测试路径，解析出通过该路径所有的约束条件，并对所有约束条件进行化简。

3)**约束条件表达式：**变量及约束条件全部提取出来之后，可利用工具自动完成约束条件表达式的求解。

4)**测试数据：**该用例将会为每条测试路径约束条件表达式生成可执行的解。

（4） **变异测试执行：**生成测试用例后，需要执行测试用例进行有效性验证。包括测试数据、原始程序执行以及变异体执行三个子用例。

1) **测试数据：**变异测试不可缺少的就是测试用例。该系统测试用例采用文件导入的方式，系统通过导入的文档读取相关的测试用例。

2) **原始程序执行：**启动 BPEL 引擎，调入部署 BPEL 程序的脚本，利用读取的测试用例对 BPEL 源程序进行测试，测试结果输出到文本中。

3) **变异体执行：**将待测的变异体部署到 BPEL 环境中，利用测试用例对 BPEL 进行测试，与原始程序输出结果进行比对，记录测试状态。

（5） **测试结果分析：**对变异测试结果统计分析。包括统计测试结果和产生测试报告两个子用例。

1) **统计测试结果：**将变异测试结果进行统计，主要是统计变异体被杀死情况。

2) **产生测试报告：**根据统计结果产生测试报告，包括变异得分和故障检测率值等。

2.2 系统设计

μBPEL 的系统架构如图 3 所示。系统包括如下六个组成模块：

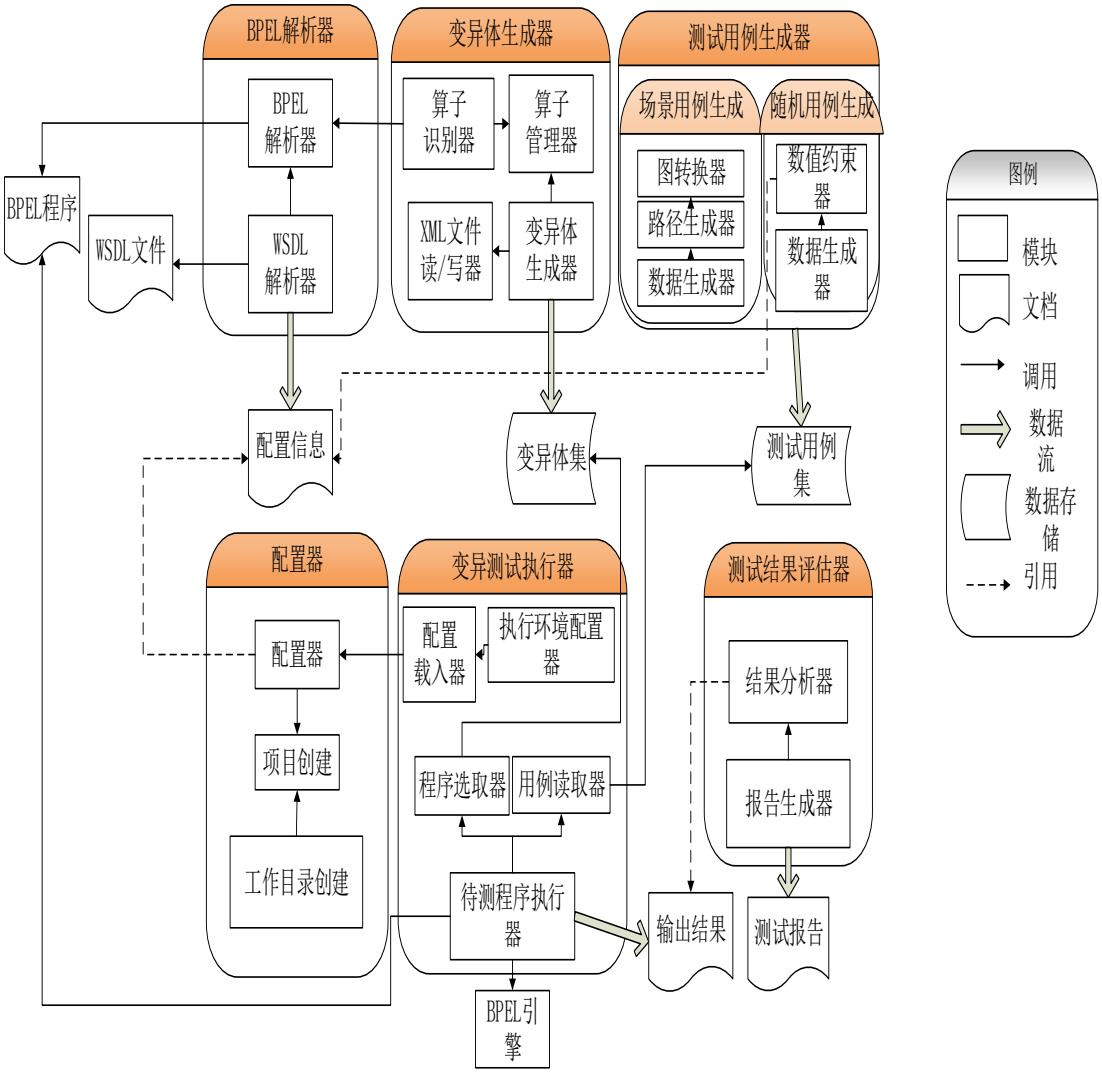

图 3 μBPEL 工具系统结构图

(1) **BPEL 解析器：**负责完成 BPEL 程序的读取，相应 WSDL 文件解析。

1) **BPEL 解析器：**读入 BPEL 文件，获取文件路径，并定义解析接口。

2) **WSDL 解析器：**解析相应 WSDL 文件，获取服务描述中定义的操作接口的输入向量格，操作名称、服务端口号等信息。其中，“格式”指输入、输出向量包含的分量名称与数据类型。

(2) **变异体自动生成器：**该模块的输入是 BPEL 程序，输出是变异体，包括四个子模块。

1) **算子识别器：**定义各变异算子的匹配与操作处理方法。该模块是系统的核心，每个变异算子均定义两个相关类，一个负责匹配 BPEL 程序，获取可以变异的节点，一个负责对相应节点进行变异处理，并将变异体写

出。

2) **算子管理器：**对变异算子的转换规则进行管理。

3) **变异体生成器：**负责生成一阶和二阶变异体。根据记录的可变异位置信息，调用相应变异算子处理方法，生成一阶或二阶变异体，并将变异体文件写出。该模块采用外观模式，通过一个外观类，将请求传递给一阶或二阶变异体生成子系统，实现客户端和子系统之间的松耦合。

4) **XML 文件读/写器：**负责读入和输出 BPEL 程序文件。BPEL 文件以 XML 形式实现，通过 DOM4J XML 文件解析工具读入 BPEL 文件，将其解析为 DOM 树形结构的 Document 对象，便于对其进行查找和修改。Document 树中的节点（Node）类型可分为元素（Element）、属性（Attribute）、命名空间（Namespace）、文本（Text）等。同时，通过继承并重写 DOM4J 解析工具自带的 XMLWriter 类，可以以基于 XML 的格式输出生成的变异体文件。

(3) **测试用例生成器：**该模块输入是 BPEL 原始文件，输出期望的测试用例。主要由如下两种用例生成方法组成：

1) **场景用例生成：**解析 BPEL 程序，转换为图模型，基于一定覆盖准则完成测试场景和数据的生成。

a) 图转换器：解析原始 BPEL 程序；提取出节点与边的信息；转换为图结构。

b) 路径生成器：进一步对图结构进行处理，变换为扩展的二叉树；遍历二叉树，生成满足一定覆盖准备的测试路径。

c) 数据生成器：负责解析 WSDL 文档；提取生成测试路径的条件表达式；利用开源 Z3 工具对每条测试路径的条件表达式进行求解，生成满足约束表达式的可行解。

2) **随机用例生成：**解析 WSDL 文档；输入约束条件；生成随机的测试用例数据。

a) 数值约束器：读入配置信息文档，获得 BPEL 服务的输入向量格式，包括输入变量及类型。读入用户指定的变量约束范围及测试用例个数。

b) 数据生成器：获取数值约束器的内容，随机生成满足数量的测试用例。

(4) **变异测试执行器：**该模块的输入是测试用例、BPEL 原始文件以及变异体，输出是变异体的测试结果，在变异测试执行的时候需要启动 BPEL 引擎，该

模块包括三个子模块。

1) **配置载入器：**读取配置信息，主要是读取用户写入的 configuration.xml 文件。
2) **执行环境配置器：**根据配置信息，配置执行环境，包括启动 tomcat、部署、反部署服务配置。
3) **程序选取器：**依次读入原始程序、变异体程序文件。
4) **用例读取器：**用户输入要执行的测试用例集。
5) **待测程序执行器：**调用 BPEL 引擎依次对原始程序、变异体运行测试用例，结果保存在 txt 文件中。

(5) **测试结果评估器：**该模块的输入是变异体测试结果，输出是测试报告，包括两个子模块。

1) **结果分析器：**读入原始程序与变异体程序测试结果文件，逐行对结果进行比对。
2) **报告生成器：**统计结果，计算变异得分，生成报告信息，包括被测试程序名字，变异体数，被杀死变异体数，变异得分等信息。

(6) **配置器：**该模块测试过程的参数配置，包含三个子模块：配置、项目创建和工作目录创建。

1) **配置器：**负责测试过程自定义参数的封装。
2) **项目创建：**负责测试项目的建立及管理。测试项目保存了待测 Web 服务的基本信息，如 WSDL URI，服务名称等。
3) **工作目录创建：**负责生成项目的工作目录，该目录保存测试过程产生的文件，如测试结果文件，测试报告文件等并将这些文件的存储位置传递给项目创建模块。

3.系统实现与演示

3.1 系统功能

μBPEL 对标准的 BPEL 程序，生成变异体，并根据生成的变异体对 BPEL 程序进行变异测试及结果分析，形成测试报告。μBPEL 主要实现了以下功能：

（1） 生成测试用例

1) 解析 BPEL 语言，识别操作。
2) 解析输入参数的类型和约束。

3) 基于测试技术或者覆盖准则生成测试用例。

（2） 生成变异体

1) 解析 BPEL 语言，识别可以变异的元素。

2) 基于分析结果，在代码中匹配相应的变异算子。

3) 根据转换规则，生成变异体

4) 支持生成一阶或者二阶变异体。

（3） 执行测试用例

1) 输入测试用例，执行原始 BPEL 程序，并记录输出结果。

2) 用同样的测试用例执行变异体，并记录它们的输出结果。

（4） 测试结果分析

1) 计算变异得分和故障检测率。

2) 总结、分析和记录测试结果。

3.2 系统演示

运用 SupplyChain 的例子来演示系统，展示系统对于 BPEL 程序变异测试过程的支持。SupplyChain 例子是一个供应链管理系统的 BPEL 例子，涉及 3 个 Web 服务。SupplyChain 例子执行流程如图 4 所示。

图 4 SupplyChain 例子执行流程

SupplyChain 接收客户的两个输入参数，分别是货物名称 name、货物数量 amount。客户(Consumer)向零售商(Retailer)发出订货请求，Retailer 根据订货单向供货商(Warehouse)发出供货请求。Warehouse 根据存货情况，反馈给 Retailer 不同结果。收到反馈后，Retailer 进行相应的处理。当 Warehouse 存货充足时，返回“yes”；Retailer 向 Shipper 发出运货请求，Shipper 收到请求后向 Retailer 发送确认消息，返回“yes”，否则返回“warehouseA canot receive the bill”；当 Warehouse 存货不足时，Retailer 向 Consumer 发送存货不足的消息，订单取消，返回“warehouseA canot receive the bill”。

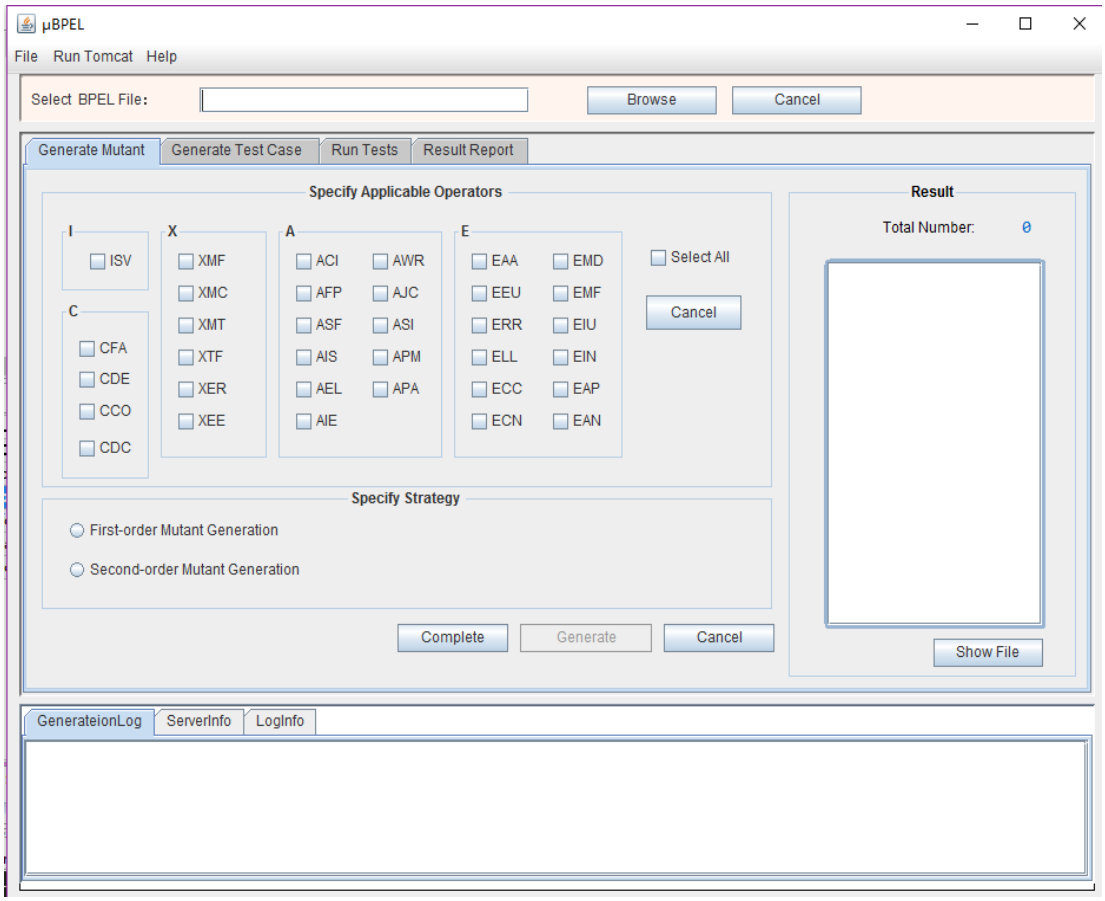

图 5 μBPEL 系统首页

如图 5 所示，系统主要由四个部分组成：菜单栏、文件选择区域、具体功能区域和日志区域。菜单部分提供重启工具、运行 tomcat、帮助三个菜单项；文件选择区域主要是显视用户选入待测的原始 BPEL 文件；具体功能区域是工具的主要操作点，包括变异体生成，测试用例生成，测试用例执行及结果分析四个部分；日志区域提供三种日志的显示:[GenerationLog]显示变异体生成时的日志记录；[ServerLog]显示 Tomcat 运行时输出日志记录；[LogInfo]显示与用户操作行为相

关的日志记录。

将按照一个 BPEL 程序的变异测试基本过程对工具进行演示。变异测试基本过程包括变异体生成，测试用例生成，测试用例执行及结果分析等步骤。

（1） **变异体生成过程演示**

1) **测试前准备**

首先，需要进行 BPEL 程序的变异测试前的准备，包括选入待测的 BPEL 程序并开启 Tomcat。开启 Tomcat 主要是为了运行 BPEL 程序提供支持。用户可以在 Configuration.xml 配置文件中自定义 Tomcat 的文件路径。当 Tomcat 开启成功后，会在界面上显示“Open Tomcat successfully”提示语。具体如图 6 所示。

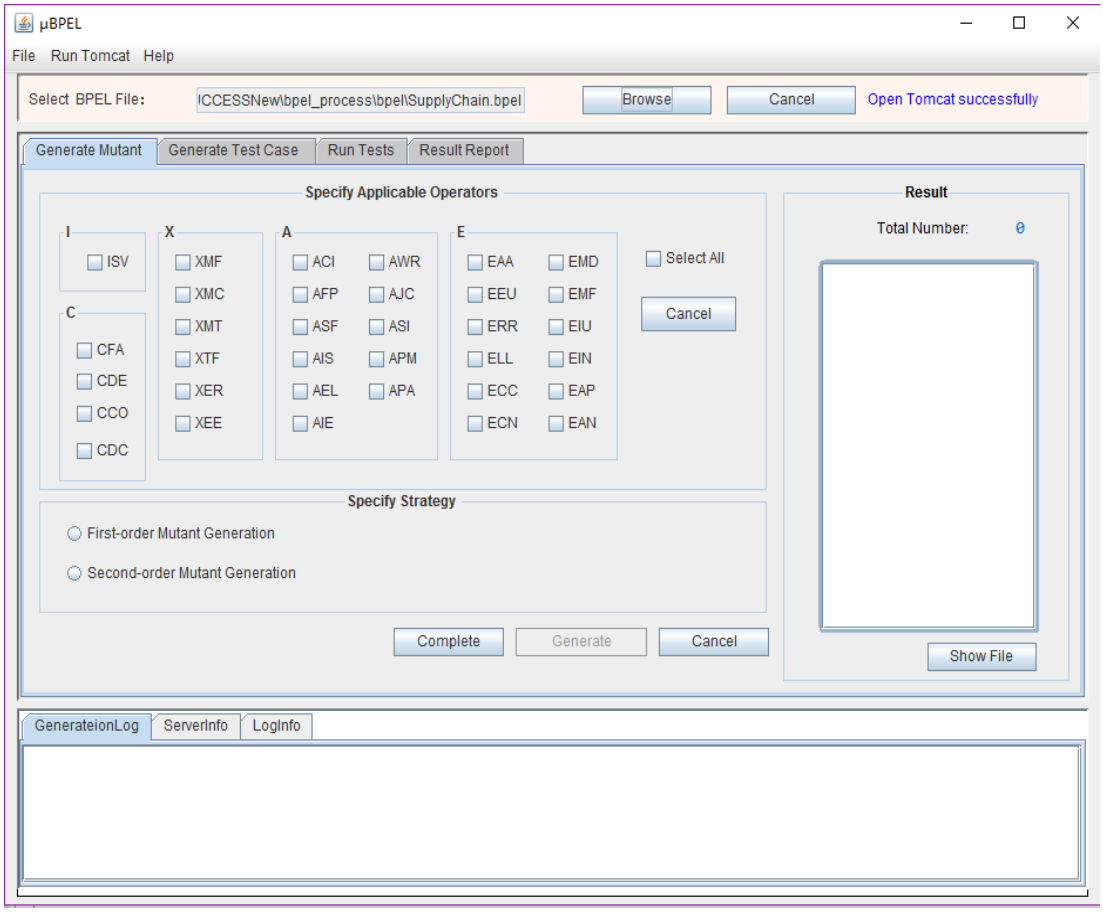

图 6 μBPEL 系统准备界面

2) **变异体生成过程**

在开启 Tomcat 和选入待测的 BPEL 程序后，可以对 BPEL 进行变异测试。通过选择［Generate Mutant］标签，进入变异体生成界面。该界面主要提供解析 BPEL 程序，识别并应用变异算子生成相应策略下的变异体集合，显示相应变异体文件等功能。

首先，进行变异体生成过程参数配置。在［Specify Applicable Operators］框中选择需要应用的变异算子(或直接点击［Select All］选择所有变异算子)，其中，鼠标指向变异算子时，界面出现相应的变异算子的转换规则描述，然后，在［Specify Strategy］框中进行策略的选择(生成一阶变异体还是二阶变异体)。

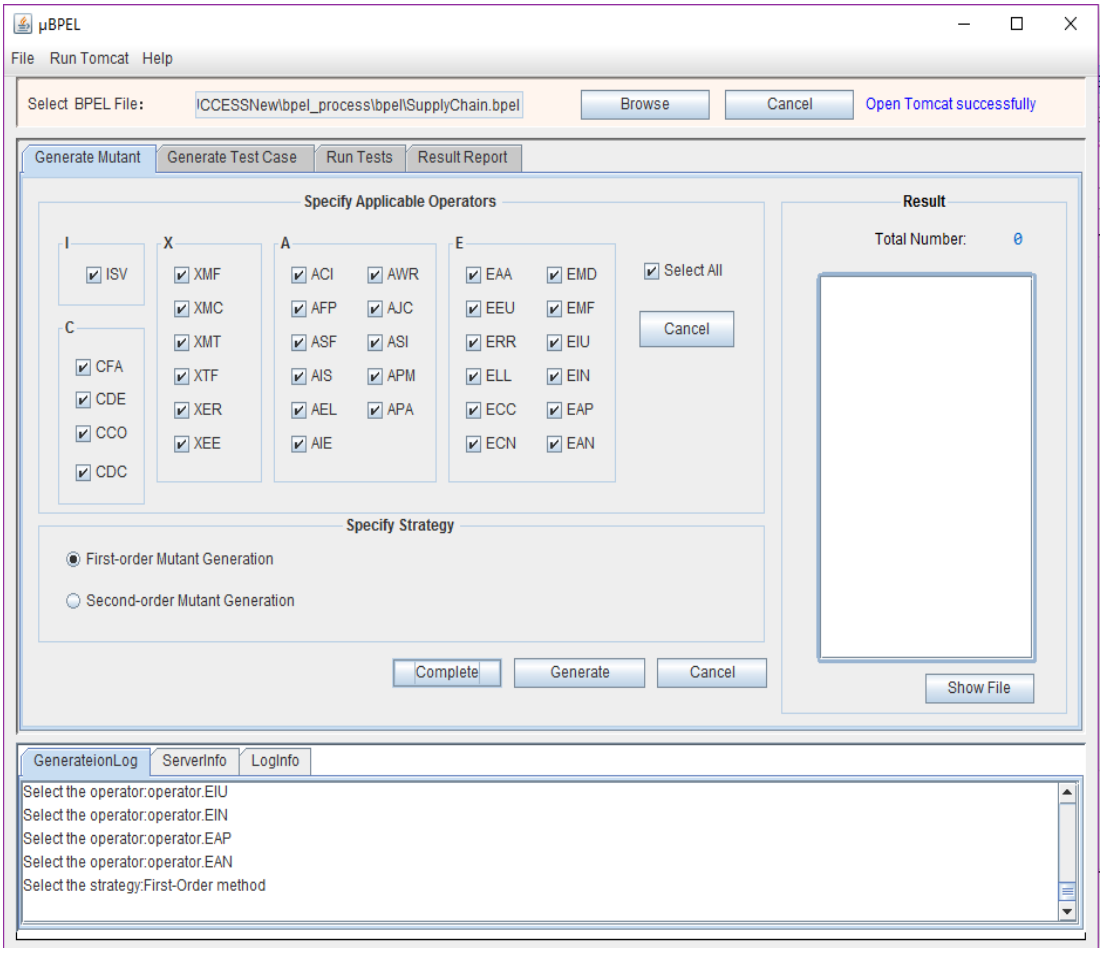

图 7 变异体生成界面

a) **一阶变异体生成过程**

一阶变异体需选择[First-order Mutant Generation]按钮,完成变异算子和策略选择后，点击[Complete]按钮，系统会将所配置的选项加载进去(图 7)。最后，点击[Generate]按钮，此时 BPEL 文件的路径及选择的变异算子将作为参数，传递到系统中。系统会识别出可以应用的变异算子，并完成相应算子的一阶变异体生成。变异体生成结果显示在[Result]框中，本例共生成 40 个一阶变异体，具体的包括 CDE_1,CDE_2 等。变异体文件由所应用的变异算子名称加“_”和数字命名(图 8)。

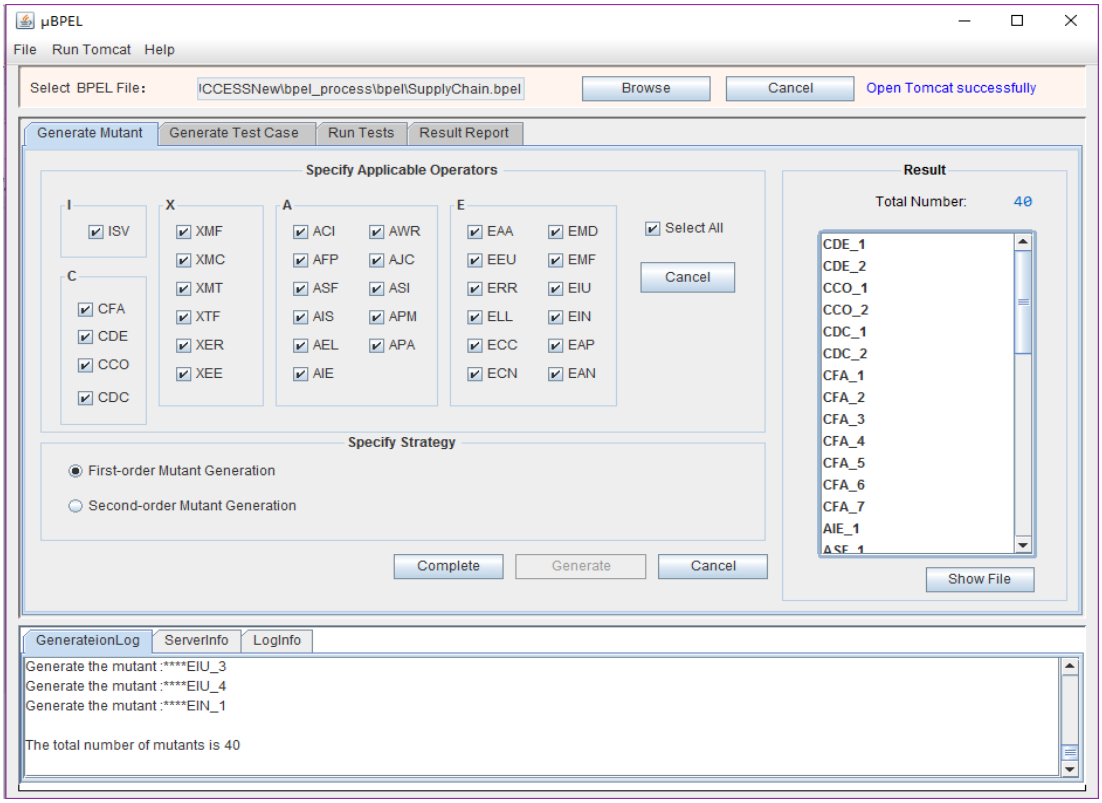

图 8 一阶变异体生成界面

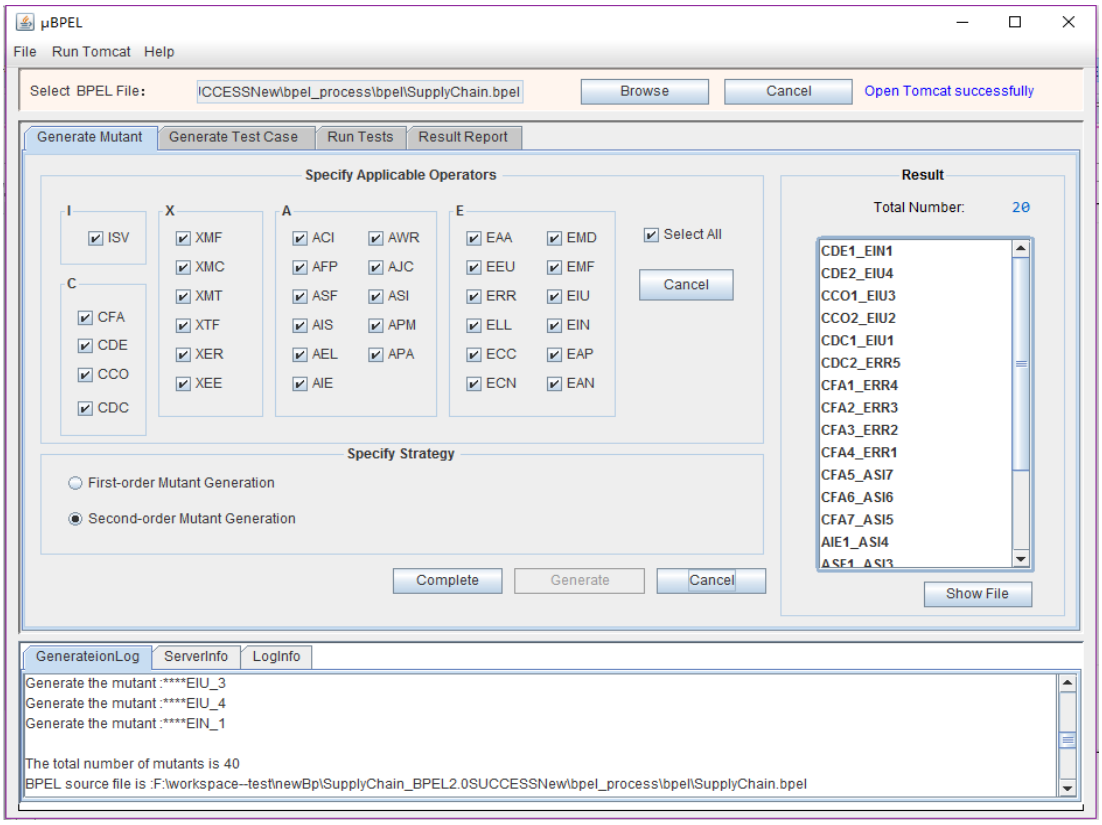

图 9 二阶变异体生成界面

b) 二阶变异体的生成过程

在[Specify Strategy]框中选择[Second-Order Mutant Generation]按钮，其他步骤同一阶变异体生成过程。系统将会运用 LastToFirst 组合算法来生成的二阶变异体。LastToFirst 组合算法是按照一阶变异体生成顺序，首尾依次组合的方式生成二阶变异体。二阶变异体生成结果如图 9 所示。

3) 变异体文件查看

为了方便查看变异体文件变异的位置和与原始程序的不同之处。系统提供一个[Mutants Display]界面展示，点击[Show File]按钮，在界面中选择所要查看的变异体，便可以清晰的看到该变异体与原始程序的异同。如图 10 所示，蓝色高亮部分显示的是变异体变异的位置。

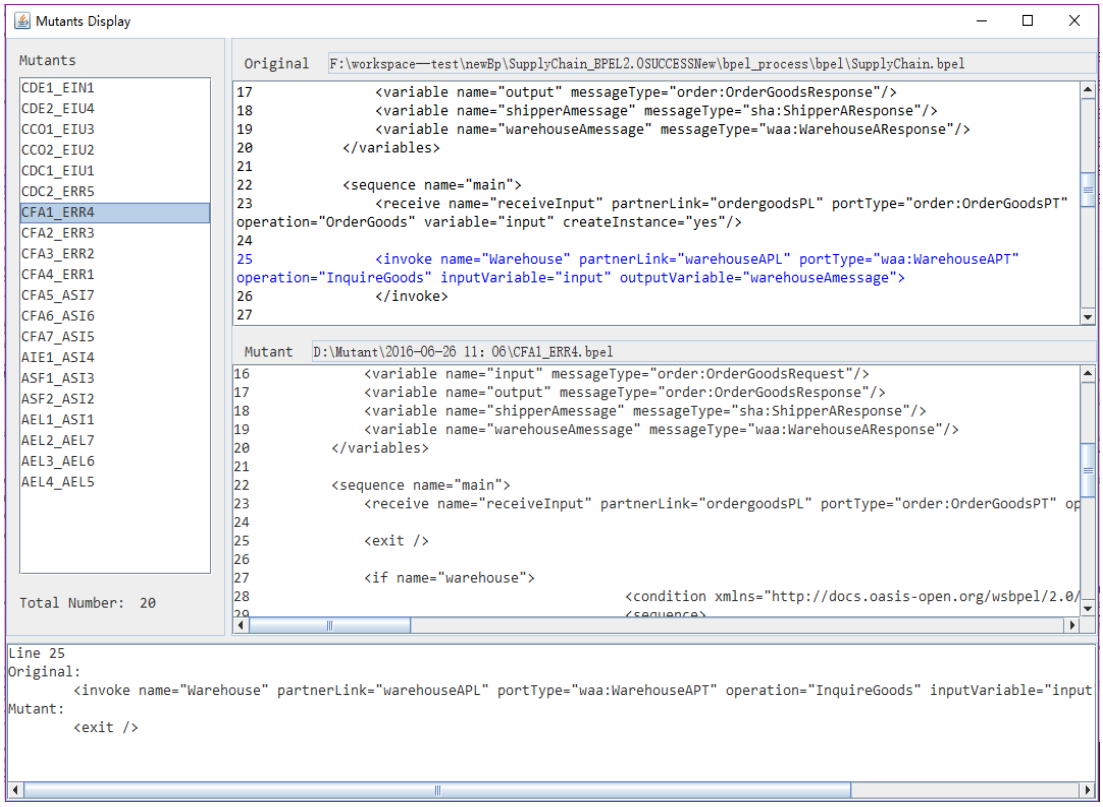

图 10 文件对比界面

（2） 测试用例生成演示

BPEL 程序的变异测试需要测试用例的支持。通过点击［Generate Test Case］菜单项，进入测试用例生成界面。该模块提供两种用例生成方法:一个是面向场景的测试用例生成方法（SOT），另一个是随机用例生成方法(RT)。具体如下：

1) 面向场景的测试用例生成演示

a) 测试路径生成过程

首先，解析器获取 SupplyChain 的 BPEL 文件，将其路径加载到系统中。点击[Generate Path]按钮，工具将会通过解析 BPEL 程序，生成不同覆盖策略下的测试路径集合。弱覆盖的测试路径集合会显示在[Weak Coverage Path]标签页下，而一般覆盖的测试路径集合会显示在[Moderate Coverage Path]标签页下。如图 11 显示，该例子在满足弱覆盖准则下，生成的两条测试路径。经过以上的步骤，就可以得到 SupplyChain 在不同测试策略下的测试路径集合。

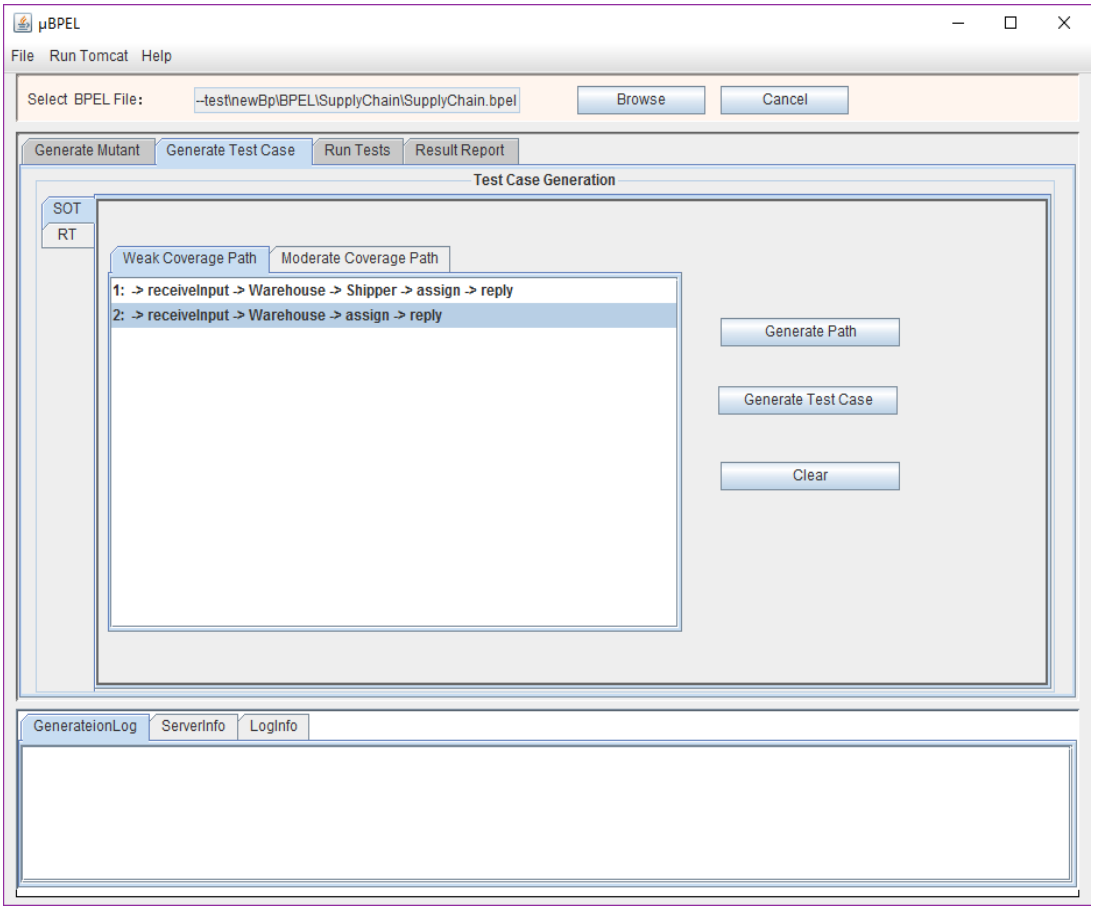

图 11 面向场景的测试路径生成界面

b) **测试数据生成过程**

得到测试路径集合，便可以针对每条测试路径完成测试数据的生成。点击[Generate Test Case]按钮，会提示“Please choose one path below”提示框。在测试路径集中的选择任一条测试路径，将会显示该条路径的测试数据生成界面，如图 12 所示。界面显示该条路径的条件表达式等基本信息。依次点击[Parse]-[Resolve]-[Extract]按钮，系统将解析条件表达式，并将条件表达式中重复的变量进行处理，得到最简化的约束条件表达式。在界面中输入预期结果与测试用例数据组数，点击[Generate]按钮，系统将会调用 Z3 工具对条件表达式进行求

解，生成满足条件的可行解并将结果保存在所选择的文件中。

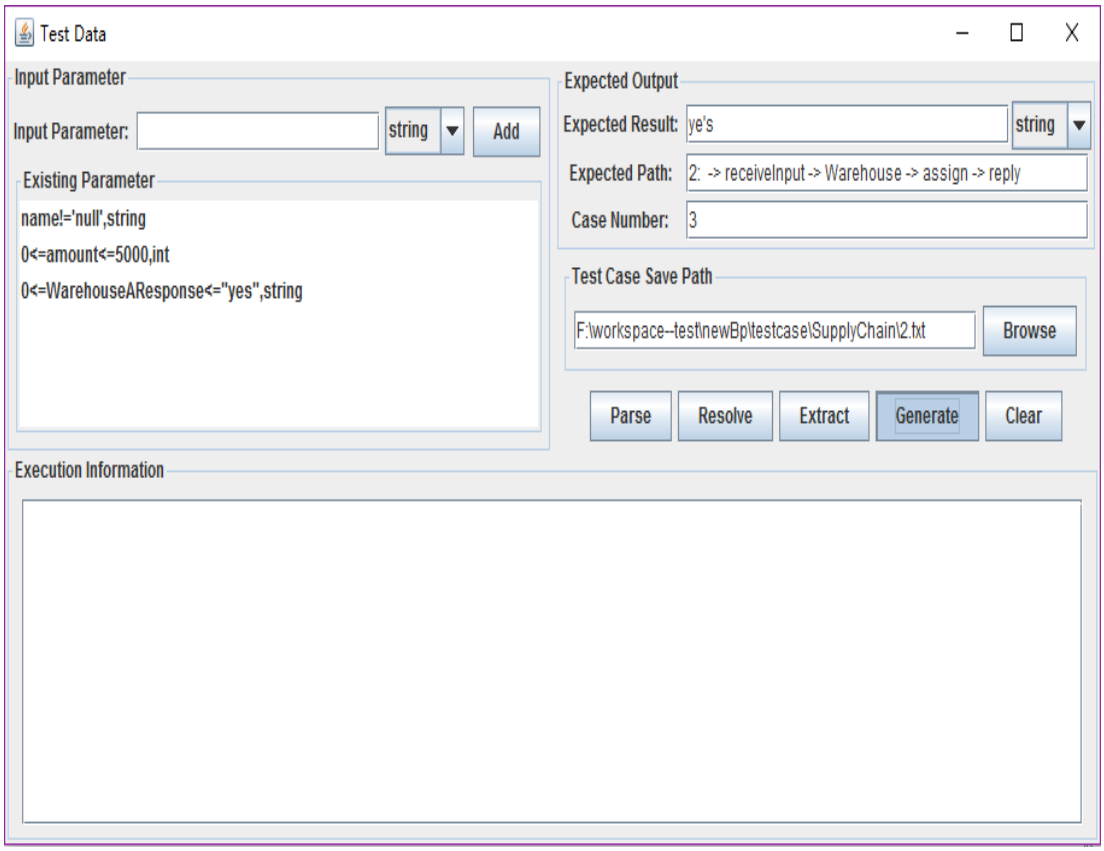

图 12 测试数据生成界面

2) **随机用例生成方法演示**

点击［RT］菜单项，进入随机测试数据生成界面，该界面主要是解析 WSDL 分析其中包含的变量及类型，将要解析的 WSDL 加载到系统中，系统会分析该 WSDL 中对应的变量及其类型，然后将确定其范围，在 Input 中显示该 WSDL 涉及到的变量。同样可以在右侧输入框中输入所需测试用例数目。然后点击[Generate]按钮，则会将生成结果保存到文本文档中。如 SupplyChain 中 OrderGoodsRequest.wsdl 对应的解析结果如图 13 所示。在输入需要产生的测试数据数目和测试数据输出路径后，点击[Generate]按钮，将会随机的在指定的参数范围内生成指定数目的测试数据集合，结果保存在文件中。

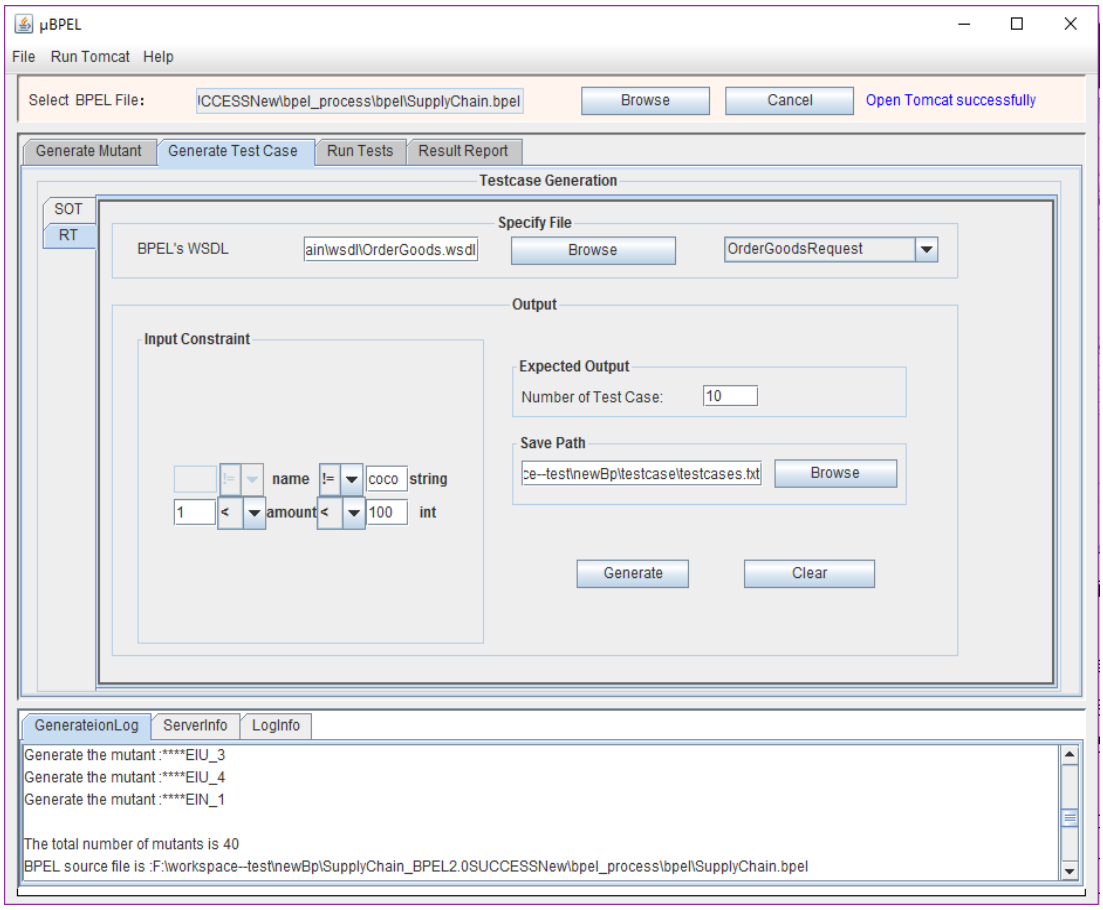

图 13 随机测试数据用例生成界面

（3） 测试执行演示

通过以上步骤，得到变异体集合和测试用例集合，可以对 BPEL 程序执行变异测试。通过[Run Tests[标签页，进入执行测试界面。在对 SupplyChain 执行测试之前,需要先将该 BPEL 文件部署到 Tomcat 上，系统通过 build.xml 脚本来完成 BPEL 的部署。然后，输入待执行的 BPEL 程序文件和所需执行的测试用例，完成基本测试环境的配置。

点击[Run Test Cases]按钮，系统将会自动获取该程序的输入端口等信息，发送测试用例，返回执行结果并将其保存在文件中。自此，对所有变异体都执行测试并获得了相应测试结果(图 14）。

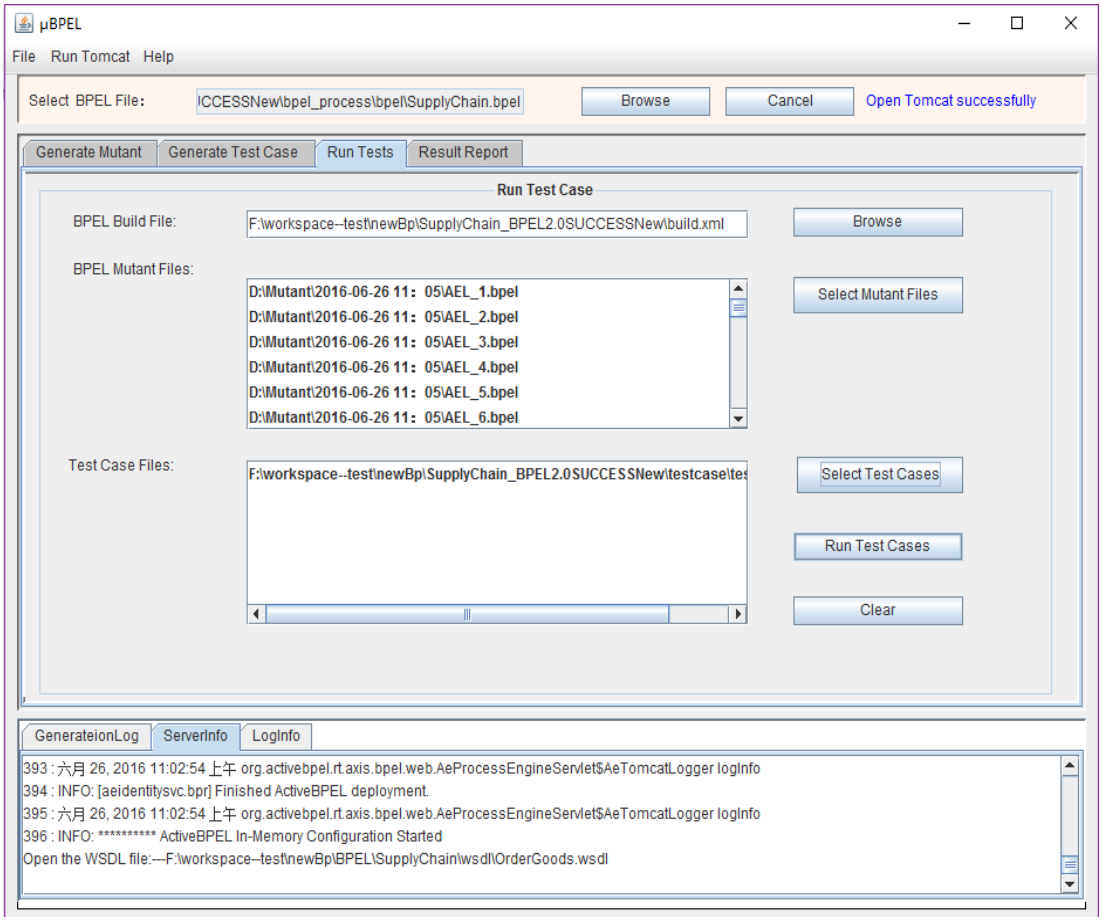

图 14 执行测试用例界面

（4） 测试结果分析演示

对所有变异体都完成测试用例的执行，还需要对其测试结果进行评估。通过点击[Result Report]菜单项，进入结果分析界面。

通过[Browse]按钮选择原始程序测试用例结果和[Select Mutatnt Output]按钮选取所有变异体的测试用例执行结果，单击[Run]按钮便可将变异体输出结果与原始程序测试结果进行比对，输出比对结果。

系统会对各变异体的故障检测率进行统计并一一显示在界面上，测试状态会在[Status]的项中显示，“F”代表该条测试用例杀死该变异体。[FinalStatus]状态栏则表明该变异体最终是否被杀死:状态为“SUCCESS”则表明该变异体没被杀死；状态为“KILLED”表明该变异体被杀死。在[Result]栏中显示的是该 BPEL 实例的测试报告，本 BPEL 程序的被杀死的变异体数量为 40 个、总变异体数量共 40 个、变异得分为 100%，表明该实例的变异体被这组测试用例全部杀死。

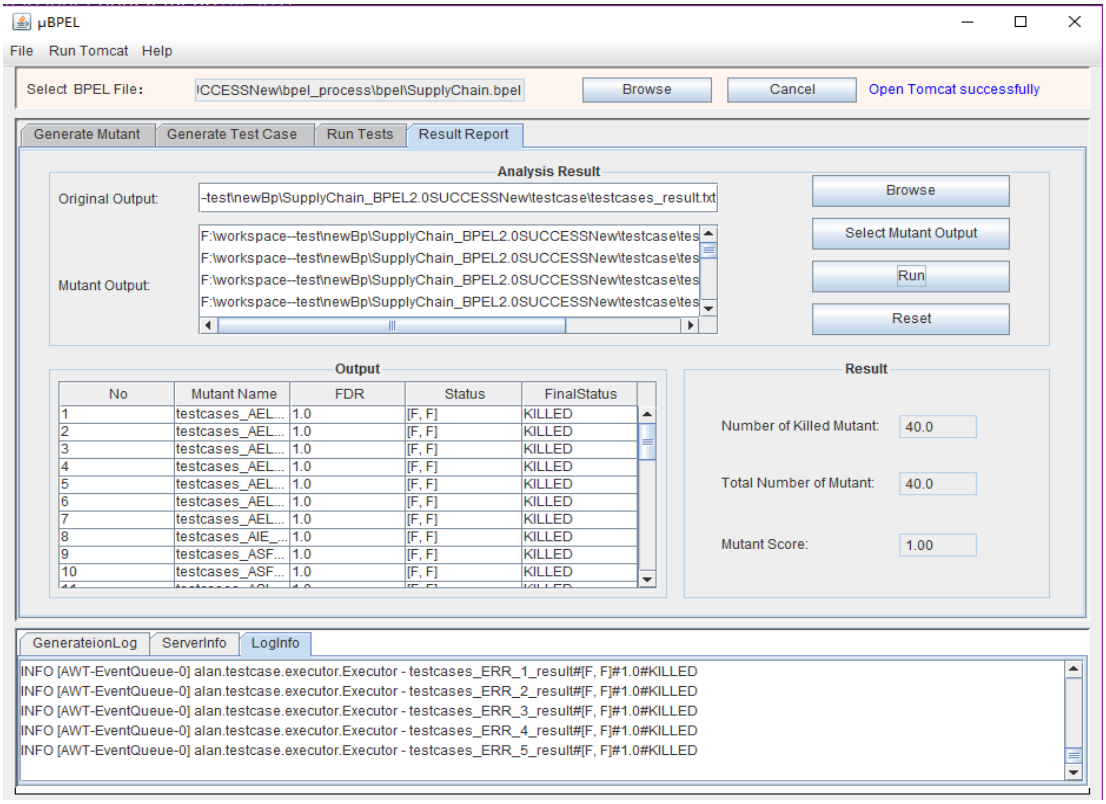

图 15 测试结果分析界面

（5） 工具帮助界面演示

工具提供变异算子详细介绍，其在[Help]界面，如图 16 所示。该界面列出针对 BPEL 程序可以应用的全部 34 种变异算子名称及其解释。

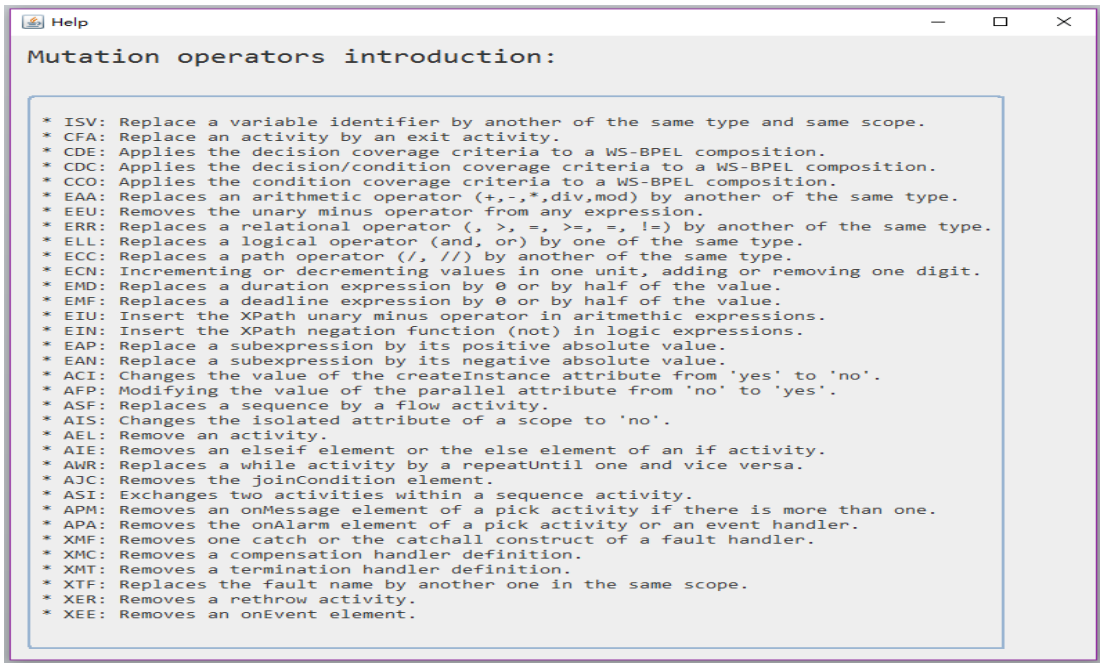

图 16 系统帮助界面