开发 EROP 编译器——一种用于智能合约规范的语言：经验报告

Adrian Delchev、Ioannis Sfyrakis[1], 和 Ellis Solaiman[1]

纽卡斯尔大学计算机学院，英国 ioannis.sfyrakis@newcastle.ac.uk，ellis.solaiman@newcastle.ac.uk

摘要。智能合约是对标准纸质合同的转换，可由合同管理系统强制执行并运行。从较高的抽象层次来看，合同只是一份描述签约方在不同情形下应如何行事的文件；然而，将典型的纸质合同转换为电子对应形式，事实证明既耗时又困难。多年来，研究一直聚焦于开发一种能够以简单短语和定义捕捉合同核心内容的语言。EROP（事件、权利、义务、禁止事项）是一种合同规范语言，它将合同拆分为若干组事件、权利、义务和禁止事项。

关键词：智能合约、区块链、监控、执行、链上、链下、物联网、隐私、信任。

1 引言

无论是企业还是学术界，都需要自动化智能合约合规监控流程的新方法 [1] [2]。设计良好的合同管理系统可以显著减少合同合规监控中的人工参与，并验证合作伙伴的业务操作是否遵守正在执行的合同条款。合同管理系统需要监控合同当事方之间的交互，以确保其公司的运营和活动得到妥善开展，并符合合同条款。纽卡斯尔大学开发的 CCC（合同合规检查器）是一项公正的第三方监控服务 [3] [4] ，就是此类服务的一个例子。该系统采用一种名为 EROP、用于编写智能合约的概念语言进行设计。EROP 是一种合同定义语言，使用 JBoss Rules（有时也称为 Drools）进行规则管理。EROP 代表事件、权利、义务和禁止事项 [5]。

CCC 的实现依赖于 EROP 本体——一组用于对合作伙伴之间业务操作的执行进行建模，并推理其动作是否符合合规性要求的概念与关系。EROP 本体

以 JAVA 实现为 Drools 引擎的扩展，从而能够将 EROP 规范语言更好、更直接地映射到具体实现。能够使用 EROP 语言表达合同，有助于扩大用户群体，因为与直接使用扩展版 Drools 编写合同以进行监控相比，使用 EROP 表达合同所需的技术知识有限。目前，将 EROP 转换为扩展版 Drools（也称为增强型 Drools 或 AD）仍是一个手动过程，需要将 EROP 合同规范映射为相应的 AD 版本。本文的主要贡献是设计并实现一种翻译引擎，以自动完成 EROP 到 AD 的转换，并针对符合 EROP 语言规范的输入生成有效且正确的输出。我们通过比较此前经过确认和验证的 EROP 到 AD 的手动翻译结果，来衡量该方案的有效性。我们的方案采用模块化结构，封装所需功能，便于今后按需维护和扩展。用于将 EROP 映射到 AD 的翻译引擎无需手动翻译，有望显著减少与翻译和映射相关的错误，并更有效地利用将纸面合同转化为对应电子智能合约所耗费的时间。

2 智能合约：背景

2.1 合约规范语言

合同在社会中无处不在，从日常生活中被视为理所当然的简单口头约定，到经过正式订立和公证、对我们的生活产生深远影响的文件，皆是如此。合同如今被广泛使用，这是不争的事实，其根源可追溯至古代社会 [6]。电子商务的发展大幅增加了组织可能参与的合同数量，导致组织难以跟上电子市场的需求。订立合同需要投入大量资源和精力，从聘用专业人员正式说明并核验合约参数，到当事方之间进行协商以及与业务部门协调，都是如此。电子化智能合约旨在实现合约订立和执行流程的自动化，同时降低开发成本。为此，必须使用一种表达能力足够强的语言来描述合同，使其能够涵盖合同的全部内容，同时消除任何歧义 [7] [8] [9] [4] [10]。为了使合同能够由任何类型的合约合规系统进行监控和执行，必须使用一种能够描述合同需求的语言对其进行形式化规范，包括法律要求、条款和内部政策，以及相关当事方及其动作。鉴于合同监控和合规服务的预期结果，是在尽量减少人工干预的情况下实现合同的自动化和执行，因此描述合同的语言必须

精确且没有歧义，从而避免需要人工解决冲突。自区块链技术兴起以来，随着比特币、以太坊、Hyperledger 等技术的发展，智能合约研究取得了显著进展。事实上，智能合约已被发现可用于多种应用领域，包括教育 [11] [12] [13] [14], 物联网[15] [16], 和服务级别协议（SLA） [17] [18] [19] [20] [21]。然而，智能合约的概念早在区块链发展之前便已存在，并可在多种集中式、分布式和混合式架构上实现 [22] [23] [24]。智能合约与区块链研究涵盖多个领域，包括性能 [25] 和仿真 [26] [27]。

拟议的合同表示语言 DocLog [28] 引入了法律咨询系统这一概念。这类系统能够针对交换的消息提供法律咨询，尽管其能力不如经验丰富的法律专业人士。DocLog 采用三层结构，结合数据、文本和语义导向的方法来交换合约条款。数据层旨在以便于事务处理系统轻松、高效处理的方式提供合同数据。文本方法则通过自然语言层呈现，力求以人类用户易于理解的方式表达合约条款。该语言使用 XML 对合同内容进行结构化，从而支持单个条款、子条款、章节和子章节，并支持使用版本与审批管理系统。尽管 DocLog 确实提供了一种相对易于人类阅读的方式来记录合约规范，但它无法监控所记录的合同，也不允许对合同中记录的属性进行任何操作。此外，使用 DocLog 语言无法记录合约规范中的若干重要方面。时间因素，例如买方必须在下单后不迟于 5 天付款，以及合同中的特殊情况（即规定发生偏离合同常规情况时应如何处理），在 DocLog 的原始版本中均未体现。尽管如此，该语言的底层架构仍揭示了不同语言层之间的通信方式，以及如何使用 EDI 转换器 [29]在各层之间进行映射。

所提出的管理架构 [30] 支持电子合同的管理，并定义了底层合同在商业合作关系中使用的本体。相关内容分为四个步骤：1）根据可能到达的系统状态，对合约目标进行基于一致性的离线验证，并评估其可实现性。2）定义并编译特定于应用的流程，以促进合同的执行，例如合同执行、监控、更新、终止、续约等。3）定义交互代理的角色。4）识别行动代理履行其在合作关系中各自角色所需的不同组件和服务。正如 [31] 中所述，该框架的主要重点是纠正性监控：检测合约规范遭到违反的情况，并尝试通过纠正措施加以补救；而预测性监控则是利用代理的行为预测此类违规，并采取行动以

彻底避免不良行为。实际合同采用基于 XML 的语言来描述，其架构为多层架构。该语言能够运用义务、许可和禁止事项等道义概念，表达条款、当事方、群组和动作等不同合同结构；尽管它采用相对高层次的声明式写作风格，但其可实现性尚不明确。该语言可以表示人们直观上能够接受的概念，但遗憾的是，无法将这些概念明确无误地转换为机器可处理的形式。与 DocLog 不同，该语言提供了一定程度的异常处理能力；不过，这一能力似乎有限，也不是该项目的主要关注点，易用性同样不是。

合同规范方面还有许多其他选择，但遗憾的是，大多数现有语言要么不注重易用性和实用性，要么没有可轻松集成的合同监控系统。BPEL [32] 使用事件演算来表示动作及其相应的影响，其目标并非商业合同，而是规范 Web 服务交互。与本文介绍的其他语言相比，它采用的声明式方法较少。该语言支持异常处理，但对异常情况的恢复能力有限；最大的问题似乎是易用性。对于非技术人员而言，这种抽象表示法可能过于晦涩，难以用于商业目的。Heimdahl [33] 是一个用于监控义务策略的平台，并使用 xSPL [34] 来规范这些策略。xSPL 的语法是声明式的，但其中一些语言构造不够直观，没有技术背景的人可能难以理解。此外，似乎无法直接进行异常处理。

2.2 EROP、增强型 Drools 与 CCC

如 [5] 所述，EROP 是一种合同规范语言，侧重于商业合作关系的执行和业务问题解决。该语言高度依赖事件、权利、义务和禁止事项，并将传统纸质合同中的信息归纳为上述构造的集合。EROP 与其他合同规范语言的一个显著区别，在于它扩展了相关能力，能够推理并为因业务故障和技术故障而出现的意外情况提供解决方案。该语言的一大优势在于，其操作层面抽象掉了底层细节，使合同撰写者能够专注于表达合同中的业务操作。该语言的另一个卖点是易于使用，即使非技术人员也能轻松上手；此外，其本体已有实现，可用于合同合规监控。

用 EROP 编写的合同由两个章节组成：声明部分用于定义规则中使用的所有参与角色、业务操作和复合义务；规则部分则描述合同中所指定实体的操作和处理，以及各种动作和例外行为。

该语言的形式语法以及有关 EROP 语法的更多细节将在后续章节中介绍。[5] 中规定的 EROP 本体是一组概念及其在 B2B 交互领域中的关系，我们利用它对业务合作伙伴之间交互的演变进行建模，以便推理其动作是否符合协议中所声明的目标。EROP 本体的思想已实现为一组 Java 类，用于捕获这些属性及其可用操作；该实现扩展了 Drools 规则引擎提供的规则语言，增加了各种不同的构造，用于推理和处理合同中业务合作伙伴的操作。本体的实现（也称为增强型 Drools）比 EROP 更不抽象、可读性更低，且风格上更接近 Java。它还需要一些额外代码，以便实现本体正常运行所需的便利性和维护工作，但这些代码对于使用 EROP 初始化合同的人类读者而言并非必要。EROP 语言可以完全映射到增强型 Drools，因此能够实现语言到语言的直接映射。从这个意义上说，创建精确且形式化的 EROP 语言语法的问题，就转化为翻译 EROP

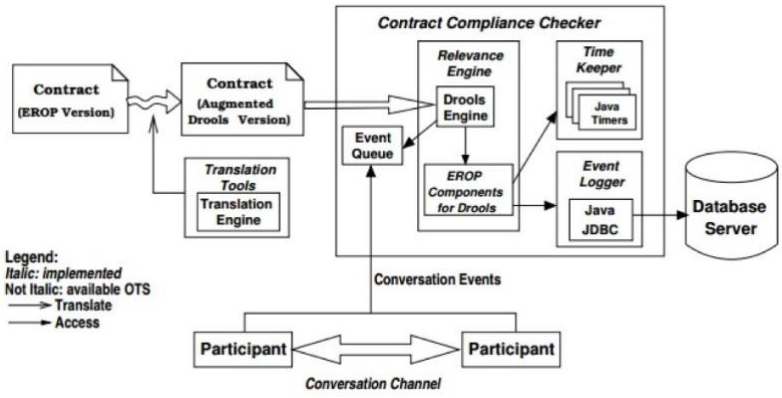


图 1. CCC 架构的表示 [5]

增强型 Drools 是本文的主要主题，将在后续章节中详细探讨。

合同合规检查器（简称 CCC）是 [3] [4] 中介绍的合同合规监控服务。它是一个中立实体，概念上位于相互交互的当事方之间，其作用是监控参与实体之间的事件交换，并推断这些事件所涉及的业务操作是否符合指定合同。CCC 的架构如图 1 所示，但由于本文的主题并非 CCC 的内部工作机制，因此不再进一步讨论。对于

本项目，需要注意的是，所有以 EROP 或增强型 Drools 表达的智能合约都可以上传至 CCC，以进行合同合规性监控。

2.3 EROP 到增强型 Drools 的转换

如上一节所述，EROP 直接映射到增强型 Drools，因为前者派生自后者。在这种情况下，将 EROP 转换为增强型 Drools，本质上就是将两种语言之间的映射自动化。下一节将详细讨论并回顾大多数类型转换中常用的技术；这里我们介绍一些更具针对性的可用技术。

如 [30], 中所述，将以自然语言或接近自然语言的格式表达的规则转换为规则标准，可以通过两步映射来完成：先将提取出的自然语言映射到 Java Bean，后者作为源语言与 JBoss Drools 生产规则之间转换的中间层。该转换技术采用一种简单直接的动词和名词概念语法，用以捕获源语言的参数，然后再将其映射为规则。尽管这篇论文描述的是自然语言或接近自然语言的映射，由于源语言缺少规则结构，与 EROP 到 AD 的映射相比，这种映射需要付出更多努力，但值得注意的是，论文使用 Java Bean 作为中间层，以捕获构建可执行规则所需的信息。[35] 的作者讨论了 Drools 和 ILOG Jrules 等面向对象规则系统所采用的方法。以 Drools 为例，这类系统构建于 Java 词汇表之上；Java Bean 被用作事实，表示规则所针对的领域及其在用户应用程序中的词汇表。规则通过导入声明使用不同的词汇表，该声明在规则文件中指定。论文介绍了如何利用 Drools 语言的底层结构（例如 Bean），将规则转换为 R2ML（一种基于 XML 的规则标记语言）。尽管论文所述转换方向与 EROP 到 AD 翻译器所需的方向相反，但它确实凸显了使用 XML 结构或语言转换为 Drools 的重要性及其优势。

2.4 总结

所开展的研究表明，在所介绍的合同规范语言中，没有任何一种语言像 EROP 一样具备如此强的表达能力，也无法处理由业务或技术故障引发的特殊情况。所有介绍的语言都要求使用者具备广泛的技术背景，非技术人员使用起来可能会望而生畏。此外，其中一些语言仅仅是用于表达合同的符号表示，并没有明确的手段来监控所表达合同的合规性。EROP 与 CCC 共同构成了一个完整的

用于捕获合同要求，并对其进行监控和执行的解决方案。将 EROP 翻译为 AD 的翻译器必须基于直接映射，并且可以使用中间表示来存储其数据。正如所评述的方法中所展示的，类 JavaBeans 结构和 XML 能够实现预期目标，其他类似项目也曾使用过这些结构和技术。下一节将介绍翻译的一些基本原理，以及它们如何应用于 EROP 到 AD 翻译器的开发。

3 方法论

开发 EROP 到增强型 Drools 的翻译器，需要对当前技术和翻译技术进行广泛分析。我们的初步目标是采用渐进式方法开展开发工作，将整个项目划分为若干具体的子项目，最终整合这些子项目，交付最终解决方案。在开发初期，重点是分析并理解翻译领域的最新进展。分析揭示了开发翻译器所必需的常见翻译技术组成部分，也让我们深入了解了该解决方案的架构设计。我们还考虑了使用外部第三方库的可能性，并在本节后文对其必要性进行了更详细的评估。该解决方案使用 JAVA 编程语言开发，因为它的实现依赖已降至最低，同时能够提供完成当前任务所需的功能。翻译目标——EROP 本体的实现——同样采用 JAVA，因此选择这种语言合乎逻辑，也有助于确保纽卡斯尔大学开发的合同合规监控解决方案 [5] 保持一致。

3.1 编译器分析

一般而言，编程语言有两个用途：既作为一种记法，向机器和人描述计算；又用于正式表达程序员的意图。此外，它们还旨在弥合不同抽象层次之间的鸿沟：较高层次更容易理解且使用起来更安全，较低层次则往往效率更高、灵活性更强。更面向人的语言需要转换为计算机能够理解的语言。这种翻译软件称为编译器。典型的编译器会将从源语言到目标语言的映射分为多个阶段。通常，第一阶段是分析阶段：它将源程序拆分成若干部分，并检查这些部分是否符合源语言的语法结构。随后，这些部分将用于创建源语言的中间表示。分析阶段的职责是检测语法和语义上的合规性或不一致性。由分析阶段构建的中间表示

称为符号表；创建完成后，它会被传递到流程的下一阶段。第二阶段是综合阶段，在这一阶段，利用源语言的中间表示生成目标语言的译文。从某种意义上说，分析部分是编译器的前端，而综合部分则是后端。进一步细看，翻译过程会按一系列步骤依次执行（图 2）。

翻译过程的第一步由词法分析器执行。它的作用是读取构成源程序或文件的字符，并将其组合成有意义的序列，称为词素；随后生成包含解析信息的词元，并将这些信息用于语法分析阶段。为了说明这一过程，以下是一个 ER OP 语言形式的输入示例：

POAcceptance in buyer .rights

词法分析器会将其拆分为以下词素：POAcceptance 是一个词素，将映射为词元 (id,1)，其中 1 指向生成的符号表中 PAAcceptance 所在的位置。符号表包含值和类型等信息。in 和 . 将分别映射为词元 (in) 和 (.)，因为它们都是该语言的运算符。与 POAcceptance 类似，buyer 和 rights 将映射为 (id,2)、(id,3)。最终得到的词元映射如下：

（标识符，1）（在）（标识符，2）（.）（标识符，3）

编译过程中的下一步是语法分析，也称为语法解析。它使用词法分析器生成的词元，创建一种类似树的中间表示，用于表示源语言的语法结构。最常见的情况是，[19] 该中间表示被称为语法树，其中每个父节点表示一个操作，而父节点的子节点表示完成该操作所需的参数。根据上述将输入字符分解为词元的结果，给定输入对应的语法树如下所示：

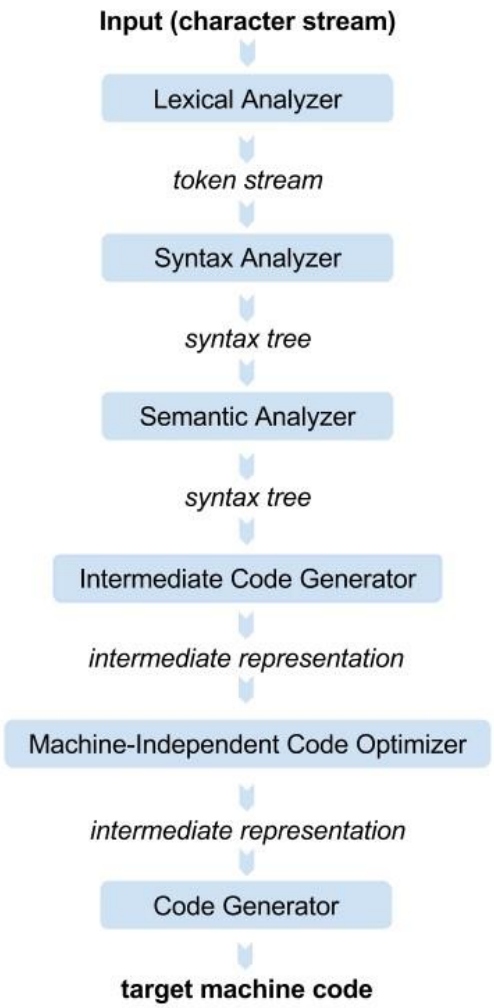


图 2. 编译器翻译的常见阶段

标记为 dot 的节点表示必须使用子节点完成该操作，并且生成的结果应作为标记为 in 的操作的右侧。该过程的下一步是语义分析器。它使用符号表中存储的信息以及语法分析树生成的信息，检查源程序在语义上是否符合语言的定义。语义分析中最重要的部分之一是类型检查，换言之，就是检查操作数是否属于指定运算符所要求的适当类型。例如，在大多数编程语言中，数组是

使用整数值进行索引；如果编译器检测到任何与预期类型不匹配的内容，就会

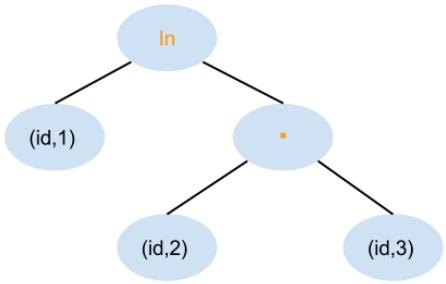


图3。语法树

应当向用户指出不一致性。对于 EROP，一个例子是业务操作的定义。如 [5] 所述，业务操作由一个以大写字母开头的通用字符串定义。因此，尝试定义业务操作时，若字符串以小写字母开头，则类型不正确，编译器必须返回错误。语义分析之后的步骤是中间代码生成。中间转换可以有多个，也可以采用多种不同形式。中间表示最重要的特性是易于生成，并且易于转换为目标语言。编译器处理流程的最后一步是代码生成，它以源程序的中间表示为输入，并将其映射到目标语言；具体采用何种方法，取决于源语言与目标语言之间的差异或相似之处。对编译器内部工作原理的详细了解，对最终解决方案的开发和设计过程极具价值。这有助于理解典型语言转换器的工作方式及其常见特征。鉴于 EROP 与增强型 Drools 在语义上相似（前者源自后者），两种语言之间的转换过程无需包含典型编译器的所有步骤，例如代码优化步骤就是多余的。此外，为了加快开发进程并避免不必要的底层错误和疏漏，可以使用能够高效实现上述部分原理的专用工具。

3.2 解析器生成器

解析器生成器是一类专门的软件开发工具，能够根据一组称为语法的规则，生成程序实现解析器所需的框架。对给定的字符输入流进行语法分析有几种不同的方法，但最重要的是自顶向下和自底向上分析算法，因为它们适用于范围最广的输入语法和上下文无关语法，也适合用于解析器生成器。解析器生成器这类工具早在计算机发展的初期就已用于创建编译器和转换器，相关实例可追溯至 1965 [36]。使用解析器生成器的两大优势，首先是开发时间：一旦熟练掌握了语法编写，借助生成的词法分析器和解析器，开发进程便能大幅加快。第二个优势是构造正确性，也就是说，生成的解析器恰好接受用于创建它的语法所规定的语言 [36]。如今，解析器生成器种类繁多，它们都采用相似的输入语法分析技术，但在语法规范的风格、用于解析输入的算法、生成的解析器文件所使用的语言等方面各不相同。如 EROP 的开发部分所述，在考虑 EROP 与增强型 Drools 之间的转换时，曾考虑使用名为 ANTLR 的解析器生成器 [5]。

ANTLR 是一种解析器生成器，应用范围广泛，包括读取、处理、执行或翻译结构化文本或二进制文件。ANTLR [37]支持动作和属性的灵活使用，这意味着可以将不同的动作定义在与语法分离的文件中，从而基本实现语法与目标语言的解耦，使面向多种语言的开发更加容易。ANTLR 还可用于生成树解析器和抽象语法树处理器。它使用扩展巴科斯-诺尔范式作为语法输入格式，并支持流行的集成开发环境。它的另一项优势是能够同时生成词法分析器和解析器，并且生成的文件采用 JAVA 格式，这使其与 AD 以及用于开发 EROP 到 AD 翻译器的语言保持一致。ANTLR 也非常流行，Twitter 使用它进行查询解析，每天处理超过 20 亿条查询；此外，它还应用于 Groovy、Hibernate、IntelliJ IDEA 等项目。

3.3 解决方案架构

在回顾了最常见的翻译技术和方法之后，EROP 到增强型 Drools 翻译器的架构设计逐渐明朗。使用解析器生成器可以加快开发，并在一定程度上复用 [5] 中定义的 EROP 形式语法。使用 ANTLR 还意味着可以涵盖典型编译器架构的前三个步骤，无需为所需功能创建独立实体。如前所述，由于目标语言与源语言相似，机器无关代码优化器并不实用，这

实质上意味着中间代码生成器可以直接与负责映射并生成最终结果的代码生成器交互。正如前一节所述，可以通过 JavaBeans 之类的实现来构建中间层。直接使用 JavaBeans 的缺点在于，它会为所有子类提供零元构造函数，这意味着这些子类可能会在无效状态下被实例化。问题源于编译器无法检测此类实例化，从而可能导致调试和跟踪变得棘手，尤其是在使用生成的解析器时。尽管如此，为了满足 EROP 到增强型 Drools 翻译器的中间代码生成层的需求，可以实现一组概念相似的 Java 类，以捕捉构成 EROP 语言的各个构建块的本质。

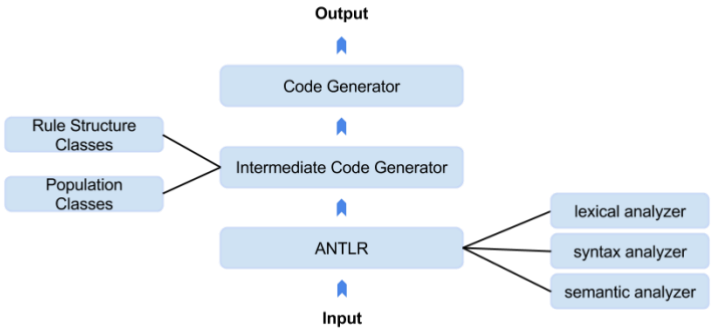


图4. EROP 到 AD 翻译器的概念架构设计

这些类的集合，以及为实现架构不同层之间的通信而需要的任何其他类，将在实现部分进行讨论。翻译器的初步概念设计如图4所示。

4 实现

4.1 ANTLR语法

ANTLR 中的语法文件本质上是一个用于规定语言语法、不同构造及其相互连接方式的文件。从较高层次的抽象来看，语法由词法规则和解析规则组成；这些规则一经定义，便会嵌入 ANTLR 生成的词法分析器和语法分析器中。词法规则以大写字母开头，这一点与解析规则不同，用于对输入进行词法化。词法规则本质上是语言的基础构建模块。而解析规则则是更复杂的规则，可以包含

规则本身，以及被视为语言基础元素的词元。如 [3], 所述，以 EROP 语言表达的合同由两部分组成：声明部分，其中定义了所有角色参与者和业务操作；规则部分，其中记录了用于合规监控的不同规则。由此可以推断，该语言的根结构是合同文件，其他所有内容都包含在该文件中。在 ANTLR 中，这可以表示为所接收输入的入口点；该入口点可以包含任意数量的子节点，具体取决于合同文件能够包含哪些内容，以及合同文件中的不同构造本身能够包含哪些内容。具体而言，图5给出了以 EROP 语言定义的合同文件结构的部分可视化表示。

如部分表示所示，声明部分可以包含一个或多个声明，而每个声明都是业务操作、角色参与者声明或复合义务声明。角色参与者声明以及标识符在

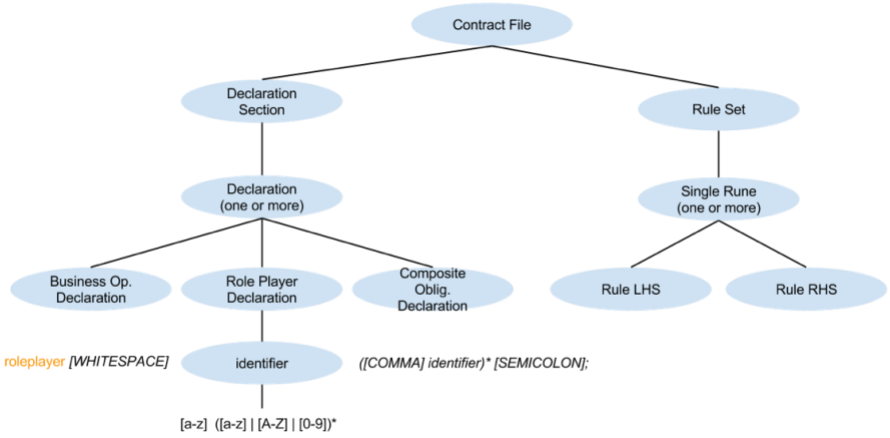


图 5. EROP 语法的部分表示。

最低层级进行定义，以展示位于规则层次结构底部的规则大致是什么样子。角色参与者声明仅由关键字 Roleplayer、其后的空格、一个或多个标识符以及末尾的分号组成。“一个或多个”量词使得可以在一行中声明多个角色参与者。标识符是一个以小写字母开头，并且可以包含大写字母和数字的字符串。

语法是 ANTLR 的重要组成部分，但仅凭语法本身并不能提供太多功能，因为相关解析器只能告知我们输入是否符合给定的语言规范。为了构建翻译器或任何类型的应用程序，解析器需要能够在遇到感兴趣的输入序列、短语或词元时触发某种

动作。幸运的是，ANTLR 提供了两种机制，允许调用动作：它会自动生成解析树监听器和访问器，以便构建语言应用程序。监听器是一种对象，当解析树遍历器发现并处理节点时，它可以响应由此触发的规则进入和退出事件；这意味着 ANTLR 会自动为任何进入或退出事件生成接口。监听器和访问器之间最根本的区别在于，监听器方法没有义务显式调用方法来遍历子节点，因此在只有输入语言的特定部分应触发事件的场景中更加灵活。另一种机制是访问器：它们必须显式启动对其子节点的访问，才能继续遍历语法树。对于 EROP 到增强型 Drools 的翻译器而言，后一种方法更为合理，因为需要对所有输入信息进行语法分析。另一种机制是访问器：它们必须显式启动对其子节点的访问，才能继续遍历语法树。对于 EROP 到增强型 Drools 的翻译器而言，后一种方法更为合理，因为需要对所有输入信息进行语法分析。

所提供的功能支持在进入语法中的某条规则时触发特定事件。不过，当这类事件发生时，仍需实现相应的特定动作。鉴于可复用性是软件开发过程中的重要一环，与其重复编写，不如复用常见概念。标识符就是一种常见且反复出现、可以复用的语法结构。它用于描述业务操作、复合义务以及角色参与者；此外，标识符不仅会出现在声明部分，也可能出现在规则集中，例如引用角色参与者和业务操作，或对其 ROP 集合进行操作。为了在实现可复用性的同时，仍能区分通用语法结构所对应的语法部分，我实现了另外两个 JAVA 类，作为 ANTLR 解析器与创建 EROP 语言中间表示的填充类之间的缓冲层。这两个类分别是 Variables Flagger 和 Variables memory。它们的作用分别是在 ANTLR 树遍历器进入不同规则时启用各种标志，然后利用这些标志判断应将已解析文件的内容存储在何处。将 ANTLR 解析器与 EROP 的中间表示之间的通信分离，符合良好的软件开发指南与实践，因为这种做法封装并抽象了判断已解析信息应存放位置所需的逻辑。此外，它也有助于测试，并推动模块化方法设计，这正是本项目的开发目标之一。

4.2 规则结构类

如分析部分所述，翻译器的中间代码生成器可以通过借鉴各种技术编写自定义的 Java类来实现。生成的 Java 类必须捕获相应

语言构造及其包含的、将中间表示链接到最终目标语言所需的任何信息。生成的类及其函数如下：

– 事件匹配条件：表示事件触发前，事件匹配必须满足的条件。其结构为「字段 运算符 值」。根据完整语言语法（见附录）的规定，字段值可以是 [3] 中指定的业务类型/结果/发起方/响应方之一。– 约束：约束类是规则中可指定的各类约束的通用集合。它可以包含以下任意一种：ROP 约束，用于捕获角色参与方的 ROP 集合中是否存在特定业务操作或复合义务；历史约束，用于根据特定事件是否存在或某指定事件发生的时间来控制规则的触发；直接时间比较和部分时间比较约束，用于对给定事件的时间戳进行额外检查；直接时间比较用于检查时间戳是否与指定时间点相同、早于或晚于该时间点；部分时间比较用于检查事件时间戳是否处于指定的小时、分钟、年份、天数或月份范围内；结果约束用于指定事件结果方面的约束，例如成功、失败、业务失败等；右侧操作表示规则的右侧，即规则中 then 与 end 之间的所有内容。它可以包含条件语句、结果或通过操作，也可以对角色参与方的 ROP 集合或业务操作的结果进行任意操作。– 条件语句：一种条件结构，用于在规则右侧捕获额外约束。它有自己的左侧和右侧。尽管它不会改变 EROP 语言的结构，却能让书写方式更加自然、高效。– 添加或移除操作：用于收集关于角色参与方 ROP 集合中各类操作的信息，例如添加或移除权利/义务/禁止事项。– 规则：规则类结构架构中的根类，包含表示和重建规则所需的全部信息。

图 6 展示了规则结构中各个类的架构层次，以及它们之间的交互方式。

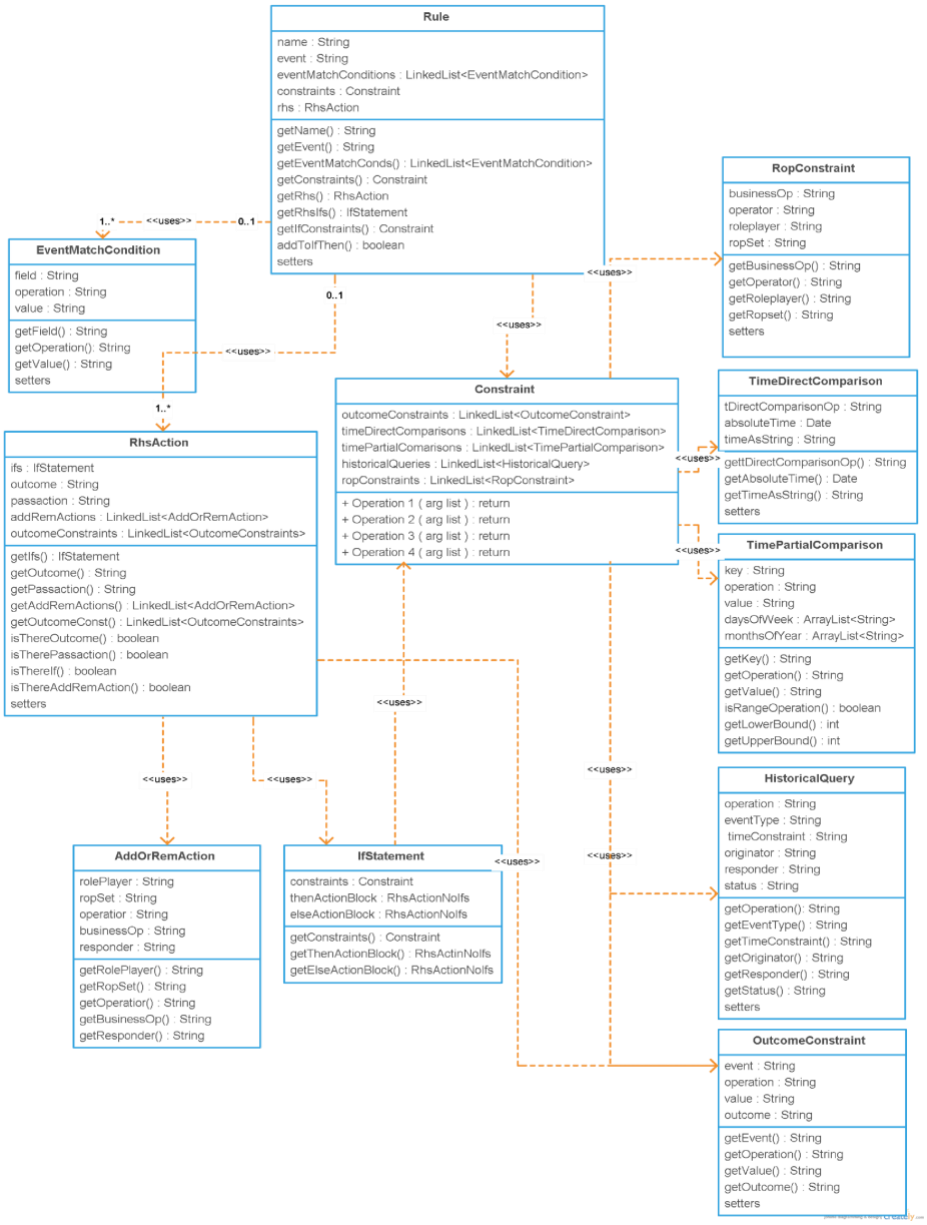

Rule
name : String
event : String
eventMatchConditions : LinkedList<EventMatchCondition>
constraints : Constraint
rhs : RhsAction
getName() : String
getEvent() : String
getEventMatchConds() : LinkedList<EventMatchCondition>
getConstraints() : Constraint
getRhs() : RhsAction
getRhsIfs() : IfStatement
getIfConstraints() : Constraint
addToIfThen() : boolean
setters
RopConstraint
businessOp : String
operator : String
roleplayer : String
ropSet : String
getBusinessOp() : String
getOperator() : String
getRoleplayer() : String
getRopset() : String
setters
EventMatchCondition
field : String
operation : String
value : String
getField() : String
getOperation(): String
getValue() : String
setters
Constraint
outcomeConstraints : LinkedList<OutcomeConstraint>
timeDirectComparisons : LinkedList<TimeDirectComparison>
timePartialComarisons : LinkedList<TimePartialComparison>
historicalQueries : LinkedList<HistoricalQuery>
ropConstraints : LinkedList<RopConstraint>
+ Operation 1 (arg list) : return
+ Operation 2 (arg list) : return
+ Operation 3 (arg list) : return
+ Operation 4 (arg list) : return
TimeDirectComparison
tDirectComparisonOp : String
absoluteTime : Date
timeAsString : String
gettDirectComparisonOp() : String
getAbsoluteTime() : Date
getTimeAsString() : String
setters
RhsAction
ifs : IfStatement
outcome : String
passaction : String
addRemActions : LinkedList<AddOrRemAction>
outcomeConstraints : LinkedList<OutcomeConstraints>
getIfs() : IfStatement
getOutcome() : String
getPassaction() : String
getAddRemActions() : LinkedList<AddOrRemAction>
getOutcomeConst() : LinkedList<OutcomeConstraints>
isThereOutcome() : boolean
isTherePassaction() : boolean
isThereIf() : boolean
isThereAddRemAction() : boolean
setters
TimePartialComparison
key : String
operation : String
value : String
daysOfWeek : ArrayList<String>
monthsOfYear : ArrayList<String>
getKey() : String
getOperation() : String
getValue() : String
isRangeOperation() : boolean
getLowerBound() : int
getUpperBound() : int
HistoricalQuery
operation : String
eventType : String
timeConstraint : String
originator : String
responder : String
status : String
getOperation(): String
getEventType() : String
getTimeConstraint() : String
getOriginator() : String
getResponder() : String
getStatus() : String
setters
AddOrRemAction
rolePlayer : String
ropSet : String
operation : String
businessOp : String
responder : String
getRolePlayer() : String
getRopSet() : String
getOperation() : String
getBusinessOp() : String
getResponder() : String
IfStatement
constraints : Constraint
thenActionBlock : RhsActionNolfs
elseActionBlock : RhsActionNolfs
getConstraints() : Constraint
getThenActionBlock() : RhsActinNolfs
getElseActionBlock() : RhsActionNolfs
OutcomeConstraint
event : String
operation : String
value : String
outcome : String
getEvent() : String
getOperation() : String
getValue() : String
getOutcome() : String
setters
<<uses>>
1..*
0..1

图6. UML图

```
rule “Rule1”
      when e matches (eventMatchConds)

      then
            if (booleanConditions)
            then
                  actionBlock1
            else
                  actionBlock2
end
```

```
rule “Rule1IfThen”
   when $e: Event (eventMatchConds)
      eval (booleanConditions)
   then
      actionBlock1
end


rule “Rule1IfElse”
   when $e: Event (eventMatchConds)
      eval ( ! booleanConditions)
   then
      actionBlock2
end
```

图7。

4.3 查找与映射

有了合适的 ANTLR语法和用于表示 EROP 的中间结构，要构建一个可用的翻译器，只剩下映射器。映射器的功能由 Translator Java类执行。顾名思义，它的作用是将规则的不同部分从中间格式转换为最终目标语言增强型 Drools。由于中间格式与目标语言相当接近，因此采用直接映射并辅以一些判断逻辑，是一种合适的翻译方式。翻译器还包含判断逻辑，用于确定源语言中的单条规则是否需要拆分为增强型 Drools 中的多条规则。这种情况最常见的原因是源语言规则结构中存在条件语句，而增强型 Drools 中并不存在条件语句。图7给出了一个简要示例，用以说明映射过程。

一般而言，EROP 中包含 if-else语句的规则会映射为增强型 Drools 中的两条规则：第一条规则将 if条件添加到 AD 的 when条件集合中，并将 then动作添加到 AD 规则的右侧。第二条规则则将否定的if条件添加到 when条件集合中，并将 else动作添加到 AD 规则的右侧。值得注意的是，只有存在 else动作块时才需要生成第二条规则；如果只有 if动作块，则无需翻译第二条规则。翻译器的最后一个组成部分是查找器，它通过一个 Java类实现，并使用一个 XML文件，其中包含 EROP 本体类的方法与 EROP 关键字之间的相应映射。该设计受到前文所述、用作中间层的各种 XML语言的启发，同时也相当实用，因为它将所有映射封装在单个文件中。如果增强型 Drools 中的方法名称发生变化，翻译器唯一需要修改的就是查找文件。

5 评估

为了严格审查并确定所开发工具的正确性，以及 EROP 语言或增强型 Drools 实现中需要进行的任何补充、改进和/或删减，我们提出了一个案例研究：从增强型 Drools 最新版本所表示的合同中提取概念，并将其作为关键要点，用于在 EROP 中表达同一合同。合同用 EROP 表示后，即可通过翻译器进行转换，再将生成的译文与以增强型 Drools 表示的原合同进行比较评估。评判翻译的两个主要方面是表达概念的能力，以及生成的译文相对于原合同的正确性。案例研究所用的合同由两个当事方签订，以下简称买方和卖方。以增强型 Drools 表示的完整原合同可在附录章节中找到。从原合同中提取的条款如下：C1：买方有权提交购买请求。提交购买请求后，在当前请求得到处理之前，买方提交任何后续请求的权利将被撤销。同时，卖方有义务接受或拒绝收到的购买请求。C2：如果在处理购买请求期间发生一个或多个业务故障，应记录第一个业务故障；之后发生的任何业务故障都应重置角色参与者的 ROP 集合。C3：收到对购买请求的拒绝后，待履行的义务即告履行，买方可以恢复发送其他购买请求的权利。C4：如果在拒绝购买请求期间发生一个或多个业务故障，应记录第一个业务故障；之后发生的任何业务故障都应重置角色参与者的 ROP 集合。C5：收到卖方对购买请求的接受后，待履行的义务即告履行，买方获得向卖方付款的新义务以及取消订单的权利。C6：如果在接受购买请求期间发生一个或多个业务故障，应记录第一个业务故障；之后发生的任何业务故障都应重置角色参与者的 ROP 集合。C7：收到付款后，买方即履行了自己的义务；买方不再负有付款义务，也不再享有取消权，并恢复提交后续购买请求的权利。C8：如果在支付购买请求款项期间发生一个或多个业务故障，应记录第一个业务故障；之后发生的任何业务故障都应重置角色参与者的 ROP 集合。C9：买方发送取消请求后，将不再负有付款义务，也不再享有提交后续取消请求的权利。C10：如果在取消购买请求期间发生一个或多个业务故障，

```
roleplayer buyer, seller;
businessoperation BuyRequest, Payment, BuyConfirm, BuyReject, Cancelation;
compoblig ReactToBuyRequest(BuyConfirm, BuyReject)
```

图 8。

```
rule "BuyRequestReceived"
      when e matches (botype == BUYREQ,originator == buyer,responder ==
store,outcome == success)
            BuyRequest in buyer.rights
      then
            buyer.rights -= BuyRequest(seller)
            seller.obligs += ReactToBuyRequest(buyer,”01-01-2016 12:00:00”)
end
```

图

```
rule "BuyRequestBnessFailure"
      when e matches (botype == BUYREQ,originator == buyer,responder ==
store,outcome == tecFail)
            BuyRequest in buyer.rights
      then
            if (BuyRequest.BizFail == false)
                  then BuyRequest.BizFail == true
                  else reset buyer
                        reset seller
            endif
end
```

应记录第一次业务故障，任何后续业务故障都应重置角色参与者的 ROP 集合。合同中定义了两个角色参与者，并规定了以下业务操作：购买请求、付款、购买确认、购买拒绝和取消。上述规定的合同条款定义了角色参与者在交互过程中 ROP 集合的变化方式。EROP 文件以合同中使用的角色参与者、业务操作和复合义务的定义开头。

合同的第二部分包含对指定角色参与者及其相互交互规则的定义。接收到购买请求时所适用的规则可以从合同的 C1 条款推导出来。该规则在成功接收到购买请求时触发。

第二条规则可以由 C2 条款推导出来，按照规定，该规则要求根据特定条件执行相应的动作。该规则可以使用 EROP 中的条件结构进行建模。

合同的第三条规则源自 C3，并在卖方拒绝买方的购买请求时触发。

```
rule "BuyRequestRejected"
      when e matches (botype == BUYREJ,originator == store,responder ==
buyer,outcome == success)
            ReactToBuyRequest in seller.obligs
      then
            seller.obligs -= ReactToBuyRequest(buyer)
end
```

图 11。

```
rule "BuyRequestRejectedFailures"
      when e matches (botype == BUYREJ,originator == store,responder ==
buyer,outcome == tecFail)
            ReactToBuyRequest in seller.obligs
      then
            if (BuyConfirm.BizFail == false)
                  then BuyConfirm.BizFail == true
                  else reset buyer
                        reset seller
            endif
end
```

图12。

```
rule "BuyRequestConfirmation"
      when e matches (botype == BUYCONF,originator == seller,responder ==
buyer,outcome == success)
            ReactToBuyRequest in buyer.obligs
      then
            seller.obligs -= ReactToBuyRequest(buyer)
            buyer.obligs -= Payment(seller)
            buyer.rights -= Cancellation(seller)
end
```

图13。

合同的第四条规则直接源自 C4，其定义与 BuyRequestBnessFailure 非常相似。当购买请求被拒绝时发生业务失败，便会触发该规则。

第五条规则定义了在收到购买请求的成功确认时会发生什么；该规则源自 C5。

第六条规则与第二条和第四条规则类似，描述了在确认过程中发生故障时会发生什么。

第七条规则源自 C7，涵盖了付款成功时发生的情况。

第八条规则描述了接收付款过程中发生特殊情况时会发生什么，该规则源自 C8。

第九条规则描述了每当收到取消请求时会发生什么。

```
rule "BuyRequestConfirmationFailuress"
      when e matches (botype == BUYCONF,originator == seller,responder ==
buyer,outcome == tecFail)
            ReactToBuyRequest in seller.obligs
      then
            if (BuyConfirm.BizFail == false)
                  then BuyConfirm.BizFail == true
                  else reset buyer
                        reset seller
            endif
end
```

图

```
rule "PaymentReceived"
      when e matches (botype == BUYPAY,originator == buyer,responder ==
store,outcome == success)
            Payment in buyer.obligs
      then
            buyer.obligs -= Payment(seller)
            buyer.rights -= Cancellation(seller)
end
```

```
rule "PaymentReceivedBFailures"
      when e matches (botype == BUYPAY,originator == buyer,responder ==
store,outcome == tecFail)
            Payment in buyer.obligs
      then
            if (Payment.BizFail == false)
                  then Payment.BizFail == true
                  else reset buyer
                        reset seller
            endif
end
```

合同的最后一条规则描述了在买入取消过程中发生特殊情况时应采取的措施。

5.1 结果与角色参与者约束

我们将首先讨论 EROP 表达增强型 Drools 最新版本中所含概念的能力，这些概念见于附录部分所展示的合同原文。合同第 2、4、6、8 和 10 条要求能够检查

某项业务动作是否已被设定为业务失败，这可以表示为图 17。图 18。

```
rule "BuyCancellation"
      when e matches (botype == BUYCANC,originator == buyer,responder ==
store,outcome == success)
            Cancelation in buyer.rights
      then
            buyer.rights -= Cancellation(seller)
            buyer.obligs -= Payment(seller)
end



rule "CancellationBFailures"
      when e matches (botype == BUYCANC,originator == buyer,responder ==
store,outcome == tecFail)
            Cancellation in buyer.rights
      then
            if (Cancelation.BizFail == false)
                  then Cancelation.BizFail == true
                  else buyer reset
                       seller reset
            endif
end



rule "Sample"
      when e matches (botype == BUYCANC,originator == buyer,responder ==
store)
      e.outcome == success
      then

end
```

图19。

一个结果约束。在 EROP 的原始版本中，结果约束确实存在，但它们针对的是事件匹配条件。例如，在 EROP 原始规范中，可以使用图19所示的语法。

其中，e 是事件名称，outcome 是事件的一个属性。这使得事件匹配条件块中可以省略某些字段，并在其后指定额外的约束。在增强型 Drools 的最新实现版本中，事件匹配块要求指定所有字段（type/originator/responder/status）。这违背了 EROP 原始版本中引入结果约束的初衷，也使其不再需要以最初引入时的形式存在；不过，在最新版本的本体中，如合同所示，结果约束可以用于业务

操作，例如检查业务操作是否失败。合同上一节所示的第 2、4、6、8 和 10 条款表达了这一点。该功能在

```
rule "Sample"
      when e matches (botype == BUYCANC)
      e.originator == buyer
      e.responder == store
      then

end
```

图 20。

为了适应 EROP 的原始版本，我们对原始结果约束进行了修改，使其具有以下语法和作用。

$$BusinessOperation.BizzFail == true/false \quad (1)$$

该结构既可用于规则的左侧，也可用于右侧；语法相同，但含义不同。用于左侧时，它位于事件匹配条件之后，与最初引入时相同。在规则左侧使用时，其作用是检查指定的业务动作是否已发生，换言之，它是一个布尔条件。在规则右侧使用时，它的作用不是布尔条件，而是用于指定该条件是否发生。换言之，在规则右侧使用时，它充当设置器。此项更改使得可以表达条款 2、4、6、8 和 10，并更新了 EROP 原始版本 [5] 中所表达的、已不再需要的结果约束版本。增强型 Drools 最新版本要求指定所有事件匹配条件字段，这也使角色参与者约束变得多余。图 20 中的语法已不再需要，可以从语言中删除。

与结果约束相同，角色参与者约束曾用于在事件匹配块之后添加额外的布尔条件，前提是省略了某些事件匹配条件字段。由于用于检查该功能的方法已从增强型 Drools 的实现中移除，因此现在已无法实现这一点。

5.2 重置 ROP 集合

增强型 Drools 中的合同还有一项原始 EROP 语言规范中未包含的功能。第 2、4、6、8 和 10 条款也体现了这一功能，它使用户能够重置给定角色参与者的 ROP 集合。在技术故障或业务故障等某些异常情况下，这对于保持角色参与者的 ROP 集合一致性是必要的。为支持该功能，我们在 EROP 的语法中添加了关键字 reset，使合同编写者能够重置给定角色参与者的 ROP 集合。与 ROP 集合操作类似，该关键字只能在规则的右侧使用。该操作示例如图 21 所示。

```
rule "CancellationBFailures"
      when e matches (botype == BUYCANC,originator == buyer,responder ==
store,outcome == tecFail)
      then
            seller reset

end
```

图 21。

5.3 案例研究评估

将上一节所示的合同输入翻译器后，会得到以下结果。增强型 Drools 中的规则文件与 EROP 中的规则文件类似，都以声明部分开头，其中声明了文件中使用的所有对象和实体。在导入 EROP 本体各类的若干 Java 语句之后，是用于声明全局标识符的部分，例如角色参与者、复合义务和业务操作。增强型 Drools 还需要 EROP 本体中其他一些类的实例，例如供规则引用的相关性引擎和事件记录器。翻译器还会自动为声明部分中指定的每个角色参与者生成 ROP 集合（符合功能需求 FR1、FR2、FR3 和 FR4）。翻译后的声明部分如下所示：

包。BuyerStoreContractEx
导入 uk . ac . ncl . erop . ∗ ;

导入 uk . ac . ncl . logging . CCCLogger ;

global RelevanceEngine engine ; global EventLogger logger ; global RolePlayer 买方 ;

global ROP 集 ropBuyer global RolePlayer 卖方 ; global ROP 集 ropSeller global 业务操作 购买请求 ; global 业务操作 付款 ; global 业务操作 购买确认 ; global 业务操作 购买拒绝 ; global 业务操作 取消 ;

翻译器正确生成了相关性引擎和事件记录器的实例，以及两个角色参与方及其对应的 ROP 集合和所有指定的业务操作（操作名称以小写字母开头，因为它们是 Java 对象，必须遵循 Java 风格规则，如 FR5 中所规定）。在增强型 Drools 中定义规则的语法与 EROP 中的相同，因为前者源自后者，其结构如下：

rule 规则名称 when 条件 then 动作 end

翻译器对前两条规则生成了如下翻译结果（图 22）。

```
rule "BuyRequestReceived"
      when $e: Event(type=="BUYREQ", originator=="buyer", responder=="store",
status=="success")
            eval(ropBuyer.matchesRights(buyRequest))
      then
            ropBuyer.removeRight(buyRequest, seller);
            BusinessOperation[] bos = {buyConfirm, buyReject};
            ropSeller.addObligation("ReactToBuyRequest", bos, buyer,”01-01-
            2016 12:00:00”));
end
```

图 22。

占位事件变量被正确转换为 $e，事件匹配条件则以增强型 Drools 格式指定。对事件

属性的约束通过 eval 关键字以及增强型 Drools 实现中的方法，在事件匹配之外施加（如 FR6 所述）。规则的右侧也得到了正确转换，对 ROP 集合的操作通过调用角色参与者所生成的 ROP 集合的方法来完成。正如预期，在复合义务的情况下，需要额外一行代码来添加新的复合义务。在上述转换中，创建了一个名为 bos 的复合义务，它由另外两个业务操作组成。根据第 2 条款推导出的第二条 EROP 规则，由于使用了条件结构，在增强型 Drools 中被转换为两条规则（图 23）。

EROP 中的单条规则被正确拆分为增强型 Drools 中的两条规则（如 FR7 所规定）：第一条规则将 if 语句中的条件添加到规则左侧，并将 then动作添加到规则右侧。这条规则还展示了经过更改的结果约束如何发挥作用：它们被正确匹配为布尔条件和设置器，并置于相应位置（左侧/规则右侧）。第二条规则通过将 if 语句中条件的否定形式添加到规则左侧，并将 then动作添加到规则右侧而生成。它还展示了新增的重置构造的转换，该构造允许合同编写者重置给定角色参与方的 ROP 集合。

其余部分的转换结果也类似，所用构造均转换正确。完整的转换结果附于附录部分，可与合同原文（同样包含在内）进行对照，以验证转换的正确性。

6 结论

所开发的解决方案能够生成翻译，但仍有一些改进空间，以增强其功能。其中包括：增加更具描述性的错误处理机制。目前，每当翻译器尝试解析文件时，都会要求输入遵循语法所规定的特定格式。如果未找到预期内容，系统会显示错误消息，指出解析器发现意外输入的文件行号和字符位置。错误消息可以进一步完善，使其描述更清晰、更便于用户理解。目前，翻译器是一个独立组件，与 CCC 分离。值得探讨将其集成到 CCC 中的成本以及所需的工作量。另一项改进是增强翻译器，使其能够将 EROP 翻译成其他规则语言。即使不加入任何建议的功能，该解决方案也能够处理所有 EROP 语言构造及其翻译。

```
rule "BuyRequestBnessFailureIfThen"
      when $e: Event(type == "buyreq",originator == "buyer",responder ==
      "store",status == "tecfail")
            eval(buyRequest.getBusinessFailure() == false)
            eval(ropBuyer.matchesRights(BuyRequest))
      then
            buyRequest.setBusinessFailure == (true)
end

rule "BuyRequestBness1stFailureIfElse"
      when $e: Event(type == "buyreq",originator == "buyer",responder ==
      "store",status == "tecfail")
            eval(!buyRequest.getBusinessFailure() == false)
            eval(ropBuyer.matchesRights(BuyRequest))
      then
            ropBuyer.reset();
            ropSeller.reset();
end
```

图 23。

参考文献

1. E. Solaiman、I. Sfyrakis 和 C. Molina-Jimenez，“合同监控服务的动态测试与部署”，载于《云计算与服务科学》第五届国际会议论文集（*CLOSER'15*），2015年。 2. ——，“基于高级模型检查器的电子合同测试”，载于 云计算与服务科学。Springer-Verlag，LNCS 第581卷，2016年。 3. C. Molina-Jimenez、S. Shrivastava 和 M. Strano，“用于检查商业交互合同合规性的模型”，*IEEE*服务计算汇刊，第PP卷，第99期，2011年。 4. E. Solaiman、I. Sfyrakis 和 C. Molina-Jimenez，“用于监控和执行电子合同的状态感知模型与架构”，载于 *IEEE*第*18*届商业信息学会议论文集（*CBI'2016*），2016年。 5. M. Strano，“用于商业交互合规性检查的合同规范”。纽卡斯尔大学，博士论文，2008年。 6. B. Nicholas，“罗马法导论”，载于 *Clarendon*，1963年。 7. C. Molina-Jimenez、S. Shrivastava、E. Solaiman 和 J. Warne，“用于运行时监控和执行的合同表示”，载于 *IEEE*国际电子商务会议（*CEC 2003*），2003年。 8. ——，“电子合同的运行时监控与执行”，《电子商务研究与应用》，第3卷，第2期，第108–125页，2004年。 9. E. Solaiman、C. Molina-Jimenez 和 S. Shrivastava，“电子合同正确性属性的模型检查”，载于 面向服务计算国际会议论文集（*ICSOC'03*）。Springer，LNCS 第2910卷，2003年，第303–318页。

10. E. Solaiman、W. Sun 和 C. Molina-Jimenez，“用于自动验证 BPMN 编舞的工具”，载于 第*12*届 *IEEE* 国际服务计算会议（*SCC*），2015年。11. B. Awaji、E. Solaiman 和 A. Albshri，“高等教育中基于区块链的应用：一项系统化映射研究”，第*5*届 *ACM* 信息与教育创新国际会议，2020年。12. B. Awaji、E. Solaiman 和 L. Marshall，“基于区块链的可信成就记录系统设计”，第*5*届 *ACM* 信息与教育创新国际会议，2020年。13. ——，“研究构建基于区块链的成就记录系统的需求”，载于 第*5*届 *ACM* 信息与教育创新国际会议，2020年。14. B. Awaji 和 E. Solaiman，“面向高等教育学生的基于区块链的可信成就记录系统的设计、实现与评估”，载于 第*14*届计算机支持教育国际会议，*CSEDU*，2022年。15. A. Noor、K. Mitra、E. Solaiman、A. Souza、D. Jha、U. Demirbaga、P. Jayaraman、N. Cacho 和 R. Ranjan，“跨多云的信息物理应用监控”，*Computers andElectrical Engineering*，2019年。16. E. Solaiman、R. Ranjan、P. P. Jayaraman 和 K. Mitra，“面向故障的物联网应用生态系统监控”，*IT Professional IEEE*，2016年。17. A. Alqahtani、Y. Li、P. Patel、E. Solaiman 和 R. Ranjan，“物联网应用的端到端服务级别协议规范”，*2018 IEEE* 高性能计算与仿真国际会议（*HPCS*），2018年。18. A. Alqahtani、E. Solaiman、P. Patel、S. Dustdar 和 R. Ranjan，“端到端物联网应用生态系统的服务级别协议规范”，*Software: Practice and Experience*，2019年。19. A. Alqahtani、D. N. Jha、P. Patel、E. Solaiman 和 R. Ranjan，“一种基于云与边缘协作的、面向 SLA 的物联网工作流活动部署方法”，载于 第一届信息物理社会系统（*CPSS*）研讨会 *2019]*, 2019。20. A. Alzubaidi、E. Solaiman、P. Patel 和 K. Mitra，“物联网背景下基于区块链的 SLA 管理”，*IT Professional IEEE*，2019年。21. A. Alzubaidi、K. Mitra、P. Patel 和 E. Solaiman，“2020年IEEE智能物联网国际会议（SmartIoT）”，*Computers and Electrical Engineering*，2020年。22. C. Molina-Jimenez、I. Sfyrakis、E. Solaiman、I. Ng、M. W. Wong、A. Chun 和 J. Crowcroft，“使用具有链上和链下组件的混合式架构实现智能合约”，*arXiv:1808.00093 [cs.SE]*, 2018。23. C. Molina-Jimenez、E. Solaiman、I. Sfyrakis、I. Ng 和 J. Crowcroft，“智能合约的链上与链下执行”，*arXiv* 预印本 *arXiv:1805.00626*，2018年5月。24. E. Solaiman、T. Wike 和 I. Sfyrakis，“采用混合链上和链下架构的智能合约的实现与评估”，*Concurrency and Computation: Practiceand Experience*，2021年。25. A. Aldweesh、M. Alharby、E. Solaiman 和 A. van Moorsel，“对智能合约进行性能基准测试以评估以太坊中的矿工激励”，*IEEE* 第*14*届欧洲可靠计算会议（*EDCC*），2018年。

26. A. Albshri、B. Awaji 和 E. Solaiman，“研究构建用于物联网应用的区块链模拟器的需求”，*2022*年*IEEE*智能物联网国际会议（*SmartIoT*），2022年。27. A. Albshri、A. Alzubaidi、B. Awaji 和 E. Solaiman，“区块链模拟器：一项系统化映射研究”，*2022*年*IEEE*服务计算国际会议（*SCC*），2022年。28. T. Yao-Hua，“DocLog：一种电子合同表示语言”，载于 第*11*届数据库与专家系统应用国际研讨会，2000年。29. “EDI 转换器”，Altova.com，访问于2023年，技术报告。30. S. Miles、N. Oren、M. Luck、S. Modgil、N. Faci、C. Holt 和 G. Vickers，“基于合同的系统的建模与管理”，载于 *AISB 2008*多智能体系统行为调控研讨会论文集，2008年。31. N. Faci、S. Modgil、N. Oren、F. Meneguzzi、S. Miles 和 M. Luck，“迈向基于智能体的合同系统监控框架”，载于 第十二届合作信息智能体会议，M. Klusch、M. Pechouˇ cek 和 A. Polleres 编。柏林，海德堡：施普林格柏林海德堡，2008年，第292–305页。32. M. Juric，“面向 Web服务的业务流程执行语言 BPEL 和 BPEL4WS，第2版”，载于 *Packt*出版社，2006年。33. P. Gama 和 P. Ferreira，“Bobligation策略：一个执行平台”，载于 第六届*IEEE*分布式系统与网络策略国际研讨会，2005年。34. C. Ribeiro、A. Zuquete、P. Ferreira 和 P. Guedes，“SPL：一种用于具有复杂约束的安全策略的访问控制语言”，载于 网络与分布式系统安全研讨会，2001年。35. G. Aiello，“从自然语言表达中推断业务规则”，载于 面向服务计算与应用（*SOCA*），*2014*年*IEEE*第七届国际会议，2014年。36. D. E. Knuth，“从左至右的语言翻译”，载于 *Information and Control 8.6*（*1965*年），1965年。37. T. Parr，“ANTLR 4 权威参考”，载于 *https://www.antlr.org/*，访问于2022年。