混合结构化编辑

面向工具的结构，面向用户的文本

Tom Beckmann[a]，Christoph Thiede[a]，Jens Lincke[a]，以及Robert Hirschfeld[a]
德国波茨坦大学哈索·普拉特纳研究所

摘要 在编程中，更好的工具往往能带来更好的结果。为此，现代编程环境提供了支持扩展的机制。工具越贴近代码，程序员就越容易将工具提供的信息对应到相关代码上。

然而，现有的扩展机制并不便于工具与文本源代码紧密集成。工具必须能够在编辑过程中跟踪程序结构，才能在正确的位置显示；但文本解析这一步使结构跟踪变得复杂。

我们提出了混合结构化编辑，这是一种支持工具构建者的方案：它既提供结构保证，也为工具用户提供熟悉且一致的文本编辑界面。
混合结构化编辑允许工具构建者声明程序必须遵循的结构约束 st
，并确保这些约束得到遵守。
我们介绍了一种实现，以及若干基于混合结构化编辑的工具案例研究，以展示其有效性。

混合结构化编辑支持安全地扩展编程环境：借助能够处理代码结构化表示的工具，并提供一致、可靠的用户体验。

ACM CCS 2012

- **软件及其工程** → 可视化语言；领域特定语言；**集成式可视化开发环境**；

关键字 结构化编辑、编程工具

《编程的艺术、科学与工程》

提交日期：2025年10月1日**发表日期：**2026年2月15日**DOI：**
10.22152/programming-journal.org/2026/11/1

© Tom Beckmann、Christoph Thiede、Jens Lincke 和 Robert Hirschfeld 本作品依据“知识共享署名 4.0 国际许可协议”授权，载于《编程的艺术、科学与工程》，第11卷第1期，2026年，第1篇文章；共31页。

混合结构化编辑

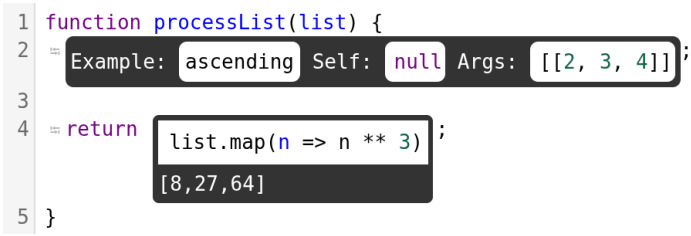


图1 一个巴比伦式编程 [39] 示例定义，它会执行所示函数，并通过监视器显示最近一次执行结果。

1 引言

集成开发环境（IDE）提供了大量支持程序员的工具，涵盖范围广泛：从可在源代码元素上显示文档的工具提示、集成式调试器，到用于静态用户界面描述的可视化设计器。这些工具让程序员能够从更广阔的视角审视程序，而不再局限于源代码文本本身所呈现的内容。

这些工具大多以轻量级装饰的形式出现在文本源代码上，或单独显示在面板或窗口中。通常，如果工具能更靠近其所指向的源代码特定部分，就能缩短相关信息之间的空间距离，从而帮助用户理解它们之间的关系 [43]。

许多研究工具都展示了这类与文本紧密集成的工具的价值，如图1所示。例如，在巴比伦式编程 [39] 中，探针直接附着在源代码中的表达式上，以显示这些表达式所具有的运行时值。交互式视觉语法 [1]展示了嵌套的视觉表达方式；用来表达程序的领域特定方面时，这些方式可以让算法更容易理解。Code Bubbles [9] 重新排列源代码文件的各个部分，以更好地传达它们之间的关系。我们将这些工具称为以代码为中心的工具。然而，创建以代码为中心的工具面临多项艰巨挑战。其中，我们着重讨论三项主要挑战：

结构跟踪 工具必须跟踪程序树中结构在编辑过程中的变化，确保即使与该结构相关的源代码发生变更，工具仍会出现在源代码中稳定且符合预期的位置。

嵌套编辑器 即使代码编辑功能深藏于工具的用户界面中，或仅涉及原始源码的一小部分，工具也必须能够提供这些功能。即使存在分离式编辑器，编辑和导航也必须始终符合用户预期。

结构化访问 工具必须能够读取源代码的静态和动态信息，将这些信息映射到相关的程序结构，并能够在保持语法正确的同时编辑这些结构。

尽管如上所述，许多工具都证明解决这些挑战是可行的，但每种工具实现都必须投入大量精力来应对这三项挑战，这可能会促使工具构建者为用户创建显示效果欠佳但实现起来更简单的工具。

作为可能的佐证，研究中发表的许多以代码为中心的工具都构建在结构化编辑器 [5, 27, 37, 41],之上，在这种编辑器中，工具处理源代码的结构和片段要简单得多。在结构化编辑器中，用户编辑的不是文本，而是直接编辑程序树的结构。因此，程序树始终有效，且结构在编辑操作过程中会保持其身份。结构化编辑器的缺点是，很难设计出让习惯文本编辑的用户感到熟悉的编辑操作 [3, 7, 44]。

本文提出了混合结构化编辑 （HybridSE）：一种通过选择性结构保证 和片段扩展普通文本编辑器的机制，以支持工具和编辑器的递归嵌套，从而应对以代码为中心的工具面临的三项挑战。HybridSE 概括了此前实现巴比伦式编程及其他增强源代码的类似图形化工具时遇到的挑战，我们将在第 2 节讨论这些挑战。为证明其可行性，我们介绍了适用于 JavaScript 代码编辑器 CodeMirror、lively4 和 Squeak/Smalltalk 的 HybridSE 实现，并展示了一系列工具案例研究来评估它。若要将 HybridSE 集成到其他编辑器中，这些编辑器必须支持在文本代码中嵌入图形元素，并实现第 5 节介绍的一个小型接口，使 HybridSE 能够负责管理图形元素的生命周期和位置。HybridSE 与语言及其范式无关——它唯一的要求是将语言解析为程序树，以便确定结构层级上的变更。

下面我们先进一步介绍这三项挑战及相关工作（第 2 节），然后举例说明如何在 HybridSE 中定义工具（第 3 节）。接着，我们将介绍 HybridSE 的基础概念（第 4 节）以及我们的参考实现（第 5 节）。最后，我们将展示一组案例研究（第 6 节），讨论我们的方法（第 7 节），并在结尾总结全文（第 8 节）。

2 背景与相关工作

本节将进一步介绍我们在第 1 节中提出的三项挑战，以及相关工作如何应对这些挑战。

2.1 跟踪程序树结构

如第 1 节所述，以代码为中心的工具面临的一项挑战是跟踪程序树的结构。以巴比伦式编程这样的工具为例：探针环绕表达式，并显示其运行时执行过程中观测到的实时值。为此，探针必须识别被探测表达式所对应的正确文本范围，并能够在用户编辑时持续跟踪该范围。

在巴比伦式编程系统中，最初通过快捷键创建探针：用户选择一个文本范围，然后按下探针快捷键。随后，系统访问程序当前的解析树，找到源代码文本中包含用户所选内容的最小表达式。然后将探针附加到该表达式上，并在其文本范围旁显示探针部件。

发生编辑时，巴比伦式编程系统现在必须确定被探测表达式所属的新文本范围。可以使用多种启发式方法来完成这一任务。首先，如果编辑发生在探针当前文本范围之前，系统可以根据添加或删除的字符数移动该文本范围。如果编辑发生在其文本范围内，则相应地扩展或缩小该范围。不过，当编辑与文本范围的起点或终点重叠时，这种简单的方法可能会产生无效的文本范围，因此我们必须重新解析文件，并选择与先前文本范围相对应、且根据此次更改调整后的最近表达式。

值得注意的是，编辑也可能引入语法错误。一旦存在语法错误，就无法再正确解析文件。由于其他语言支持工具也面临这一挑战，例如语法高亮，解析器研究提出了各种错误恢复机制，以便在存在语法错误时提供有用的输出 [12]。然而，即使采用错误恢复，也可能间歇性地无法明确我们所跟踪的表达式对应于哪个文本范围。

作为一种替代方案，结构化编辑器要求用户直接在程序树的结构层面表达编辑操作，从而消除文本编辑中的任何歧义。因此，这些结构化编辑器为工具创建提供了良好基础 [8,16, 17, 27, 40, 41]。但由于这类编辑器通常追求与文本编辑器及其语言的兼容性或相似性，因而会产生不匹配。以往研究探讨了如何让类似文本的编辑或文本语言能够在结构化编辑器中实用 [3, 31, 35, 44,45]。而本文的方法则旨在让工具能够访问结构，同时让编辑器用户能够访问文本。

在其他文本编辑器中，可用于处理结构的方式有限。例如，在语言服务器协议中， [33], 用户通过结构在文本缓冲区中的位置，或通过当前解析状态的快照来识别结构。程序树差异比较方法可以识别程序在两种状态之间发生的变更 [19, 20]。在混合结构化编辑中，我们集成了程序树差异比较，从而免去工具构建者手动比较差异和跟踪结构的复杂工作。

2.2 嵌套编辑器

Code Bubbles [9] 等研究项目展示了编程工具通过重新排列源代码来更好地适应程序员任务的可能性。然而，大多数集成开发环境只支持以单一整体视图查看源代码，最多只能在单独的视图中显示部分代码行，或显示笔记本单元格的内容 [38]。研究中的示例展示了如何组合子程序，例如 Eco [13], 、PolyJuS [36], 、Moonchild [14]，或 Engraft [22]。而我们的目标是让工具能够处理小到具有意义的单个字符的代码片段，例如单个表达式。以小片段的形式显示源文件时，会出现各种挑战。例如，在片段边界处进行导航时，可能需要跳转到视觉上相邻的片段、跳转到显示相邻文本位置的片段，或保持在当前片段内。

此外，一个片段很可能会显示程序树中某一特定节点的文本范围。因此，片段管理面临着与探针相同的问题

代码清单1 隔离跨越多个缩进行的文本范围的示意图。

```
1 // 原始完整源文件，我们希望
ma
10
spl
```

正如第 2.1 节所述，在巴比伦式编程中，当用户在片段边界处进行更改时，他们很可能会期待更改也显示在该片段中；然而，由于系统会跟踪节点的文本范围，部分更改可能不会被视为该片段的一部分。例如，考虑将一个数字改为与另一个数字相加的表达式。

```
1 2
2 2 + 3
```

当用户在输入加号之前先输入空格时，该空格可能看起来像是被忽略了，因为被跟踪的表达式（即数字）仍然只包含字符 2。

最后，当在跨越多行的片段中隔离一个表达式时，格式可能会成为问题。例如，该表达式可能位于缩进较大的位置。在片段中显示时，缩进会使除第一行以外的所有行都向右移，如列表1所示。例如，在 Code Bubbles 中，这个问题通过根据局部缩进级别自动重新格式化每个包含代码的气泡来解决。

2.3 对程序树的结构化访问

作为第1节的最终挑战，我们提出了从程序中读取静态和动态信息，以及以保持程序语法正确性的方式执行编辑。读取静态信息相对简单，因为这只需要提供一个接口，以访问和遍历已解析程序树的结构。然而，要以对附加到程序树特定节点的工具有意义的方式读取动态信息，就需要一种机制，将程序运行时收集的信息映射到感兴趣的特定节点。此外，还可能需要收集上下文信息，例如当前调用栈，以便有效利用所收集的信息。

编辑程序树时，可能会遇到数量多得令人意外的边缘情况，具体取决于语言语法。例如，修改

字符串内容时，需要遵循引号、换行符和其他特殊字符的转义规则。又或者，将表达式嵌套为二元运算的左侧时，需要检查该表达式及其父节点的结构，以确定是否必须添加括号才能得到预期的含义。

从更广泛的相关工作来看，语言工作台 [18]旨在促进新语言的创建，其中可以包括向工具提供有关语言结构的信息。另一种让编辑器能够获取程序静态信息的方法是语言服务器协议 [33](LSP)。LSP 解决了*IDE*可移植性问题 [26]，，在此问题中，每个工具都必须针对某个特定编辑器的扩展机制进行适配。借助 LSP，编辑器实现单一协议，充当语言客户端；语言支持工具也实现单一协议，充当语言服务器，从而可以在任意编辑器与语言支持服务器之间建立连接。除此之外，还有一些框架和系统专门用于促进程序结构信息的处理，例如 Moldable Tools [11]、Lorgnette [21],或 Vivide [42]。为了将运行时执行的结果映射回编辑器，源映射 [25] 是一种方法，它存储从原始源码到生成源码的映射。

3 以代码为中心的工具的定义

在本节中，我们从工具构建者的视角介绍 HybridSE 所支持的、用于定义以代码为中心的工具的框架。随后在第 4 节中，我们描述底层的 HybridSE 系统。该框架将数据流映射为由五个步骤组成的序列：从对程序结构信息的**匹配**、**提取**和**约束**，到定义用于展示这些信息的**视图**，以及定义会导致程序结构发生变更的**交互**。

为说明该框架，我们描述一种监视工具，它会报告用户所标记表达式的运行时信息。我们的监视工具应出现在所有被如下形式的特殊标记构造包裹的表达式周围：

```
1 ["__监视器",list.map(n => n **3)][1]
```

标记构造是一个数组，其第二个元素为原始表达式。该标记的第一个数组元素是一个可供我们匹配的特殊字符串。该数组表达式的求值结果为其第二个元素，因此被标记的表达式可以直接用来替代原始表达式。

我们的目标现在是用监视用户界面替换这个标记表达式，显示 list.map(n => n**3) 表达式在运行时所取的值，并隐藏标记构造，如图1所示。在源代码中包含标记，可以让我们在用户进行变更时更容易定位被监视表达式的位置。在下文的说明中，我们将按行号引用清单 2 中展示工具定义的代码清单。

(1) 程序结构匹配 首先，工具构建者定义一个查询，用于匹配程序中的结构。该查询是一个纯函数，用于返回给定对象是否

清单 2 图1中所示监视工具的声明。 1 const watchTool = { 2 query: node => { 3 const ma ex 8 } ({ e toc ret ex {ex {re

程序结构符合工具所要查找的结构。随后，HybridSE 会对编辑器中可见的所有结构运行此查询函数，并为每个匹配项实例化工具。

我们采用了先前工作中的匹配语法 [4], ，通过美元符号前缀元素指定要匹配的字面语法和变体点，而不是使用例如命令式访问节点字段的方式来判断它们是否匹配，[1] 如第 3 行所示。如果匹配失败，我们会在第 4 行返回 `null` ，以告知系统不应针对该程序结构实例化我们的工具。

(2)提取数据 接下来，工具构建者会从匹配到的结构中提取信息。概念框架提供了一个导航接口，使工具构建者能够访问结构的子级、父级和数据。此外，它还提供了一种通用机制，用于订阅结构在适用情况下所具有的运行时值。

对于上面的监视器示例，我们既在第 5 行提取了表达式结构本身，也在第 6 行提取了一个流，该流会在执行期间生成此表达式对应的运行时值。

(3)约束根据从程序结构中提取的数据，工具构建者现在可以继续声明 HybridSE 将维护的约束。在我们的

[1]如果美元符号是有效的语言语法，可以添加一个预处理步骤，将该字符替换为其他一串有效但较少使用的字符。

框架中，约束是纯函数，它们接收新的程序树以及对程序树的一组变更，并返回一个布尔值，指示这些变更是否会违反约束。

由于示例中的监视器会在其用户界面中包含状态，也就是过去值的历史记录，因此我们希望确保某项更改不会导致它偶尔消失；如果第3行中定义的查询不再匹配，就会出现这种情况。为避免这一点，我们确保查询仍适用于新树。随后，HybridSE 负责确保在对程序应用任何编辑之前，始终满足这一约束。我们将在第4.1.2节中介绍 HybridSE 对违反约束的情况作出的响应。

(4)视图工具用户界面的视图定义方式取决于该工具所托管的编辑平台。例如，我们的参考实现托管在浏览器中，因此使用浏览器的文档对象模型来定义其用户界面。

为了解决第1节中提出的**嵌套编辑器**这一挑战，HybridSE 要求实现 HybridSE 的宿主编辑器提供片段：这些片段是功能齐全的代码编辑器，这意味着它们支持错误高亮、自动补全以及宿主编辑器的所有其他功能，但只显示一个或多个连续程序结构节点的文本范围。片段可以包含工具，而这些工具又可以包含片段，从而实现编辑器的递归嵌套。从用户的角度来看，片段与工具之间的导航和选择应当是可预测的。片段还会对所显示结构的缩进级别进行规范化，以解决列表1中所述的问题；我们将在第4.2节中介绍相应的方法。

在我们的示例中，我们使用嵌入在 JavaScript 中的 HTML 定义用户界面，并使用花括号插入值，如第 14 行所示。我们在第 15 行嵌入了一个片段，用于显示监视表达式的编辑器，以及最近观察到的运行时值（如果有）。我们还在第 17 行定义了点击监视器容器时应移除该监视器。

除了像第 20 行所指定的那样替换整个匹配结构的视图之外，我们的框架还支持定义视图应插入到匹配结构之前或之后，或者在与该结构对应的文本范围中添加标记。这些标记可以添加高亮、改变字体大小，或使该范围可点击。

（5）交互 最后，我们声明视图响应用户交互时触发的回调。因此，对于监视器，我们在第 11 行声明了 removeWatch 函数。我们使用程序结构感知编程接口来编辑树，具体内容将在第4.3.3节中详细介绍。这里，我们使用其 replaceWith 方法，将调用该方法的结构替换为所提供的结构。此外，我们还指定这是用户删除工具节点的意图。根据这一规范，HybridSE 知道应忽略要求工具节点保留在树中的约束。

结果 此工具定义解决了第1节中提出的挑战：**结构跟踪** 无需明确跟踪编辑过程中的结构。HybridSE 会为工具实例化所在的节点提供稳定引用，而工具声明的约束则确保该节点保留在树中。

嵌套编辑器 该片段封装了围绕源代码某一部分构建用户界面的复杂性。

结构化访问 具备结构感知能力的访问和编辑界面，使工具构建者能够针对静态和动态信息，按照与其意图表述相匹配的抽象层次处理结构。

在下一节中，我们将介绍 HybridSE 如何实现这一概念框架 .

4 混合结构化编辑

HybridSE 实现了第 3 节所述的框架。它旨在集成到通用编程编辑器中。为了完整实现 HybridSE，宿主编辑器需要具备以下能力：

1. 与导航、撤销和编辑事件集成，2. 将编辑器组件中的文本范围替换为用户界面组件，以及3. 创建编辑器组件实例，使其显示文件中的特定文本范围，同时编辑器功能仍在整个文件的上下文中运行。HybridSE负责同步所显示的文本范围和文本内容，但其他编辑器状态（例如与 Vi 这类模态编辑器集成时的模式）必须由宿主同步。

因此，在宿主编辑器中实现 HybridSE，需集成导航和编辑事件，以跟踪结构并更新用户界面元素（第4.1节）；实现片段（第4.2节）；并实现对程序静态和动态信息的访问（第4.3节）。附录B中提供了参考实现的源代码引用。

4.1 跨编辑维护和跟踪结构

为应对第一个挑战，HybridSE将两种机制集成到编辑器实现中：第一种是结构差分机制，它使用启发式方法，在编辑过程中维护程序结构节点的身份；第二种是事务机制，用于确保工具构建者定义的约束得到维护。这两种机制共同构成了选择性结构保证：HybridSE仅在工具的约束要求时严格维护结构，否则会持续解读文本变更，以访问信息结构。

4.1.1 差分

解析程序文本会生成一棵由程序结构节点组成的树。用户对文本进行变更后，这棵树便过时了：例如，这一变更可能修改了数字的值、删除了数组中的一个成员，或引入了语法

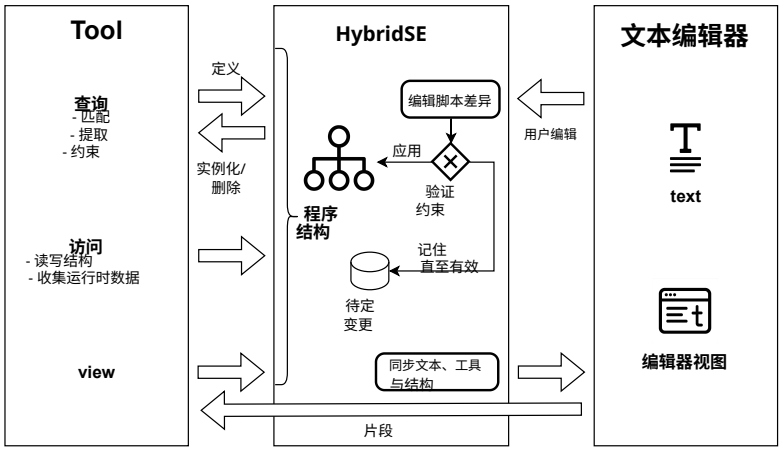


图2 HybridSE 的概念结构在工具与宿主编辑器之间架起桥梁。HybridSE 与文本编辑器集成，但向工具暴露结构。

错误，从而不再能够形成程序的有效树。因此，要找出变更后的程序树结构，我们必须再次解析文本。由于新树与旧树之间很可能存在大量重叠，增量解析器可以重用之前运行得到的解析结果，以加快新的解析过程 [15, 30, 46]。值得注意的是，这些增量解析技术并未描述树是如何变更的——它们只会加快创建新树的过程。例如，尽管 Tree-sitter 的增量解析器 [30] 会出于性能原因重用相同的对象标识，但其重用机制并非旨在尽量减少树中的变更，即使只是很小的变更，也可能重建整个子树。

由于我们的目标是保持节点的身份，使工具出现在正确的位置，并从正确的结构中提取数据，因此我们需要手动识别文本变更对树结构的影响。为此，我们改编了 truediff [19],这一结构差分算法，它采用一组启发式方法来计算程序树两个版本之间的编辑脚本。truediff 的作者评估了其启发式方法的质量，表明其与其他流行的结构差分方法相当。他们的评估针对提交级变更，而在我们的用例中，大多数差分发生在仅相隔几次交互的状态之间，因此在我们用例常见的场景中，结果可能更好。

一般来说，我们提出的系统可以与任何结构化差分方法配合使用，该方法会生成一个编辑脚本，描述必须应用于源树才能得到目标树的变更。这样的编辑脚本可以由不同粒度的操作组成。例如，truediff 定义了五种基本操作：

加载 创建一个新树节点。**附加** 将给定的树节点附加到父节点。
分离 将给定的树节点从其父节点分离。**移除** 删除一个树节点。
更新 更新树中叶节点的文本内容。

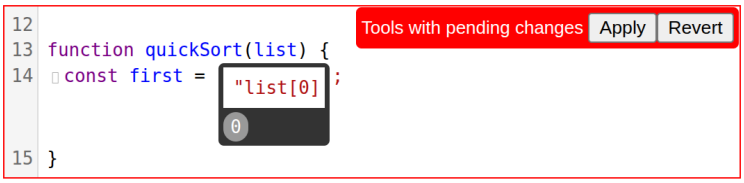


图3 用户刚刚在监视表达式的开头插入了一个引号，导致解析树无效。HybridSE 会显示通知，告知用户当前状态无效，并让用户选择通过宿主编辑器应用或还原其变更。

例如，如果用户在数字后输入加号，编辑脚本至少会包含：分离现有数字的分离操作；为二元运算符、二元运算和右操作数的占位节点加载的加载操作（这需要错误恢复解析器）；以及针对运算、操作数、运算符和数字的附加操作。计算出用于生成目标树的编辑脚本后，我们将其应用于源树。这样一来，尽管在我们的示例中数字节点在树中的位置发生了变化，它仍保持自身的身份，因为它只是作为编辑脚本的一部分被分离后又重新附加。

4.1.2 事务

差异计算步骤生成的编辑脚本会指出哪些节点被移动、更新或删除。因此，我们的工具所指定的约束只需验证编辑脚本不会造成违反这些约束的影响。例如，如果用户在示例中由清单 2 定义的监视器所跟踪的表达式中添加一个引号（如图3所示），该监视器将无法再正确解析。为防止这种情况，我们在清单 2 第9行定义了一个约束：包含用户变更的程序树中仍应匹配我们的查询。

如果编辑脚本违反此约束函数，也就是说，约束函数返回false，那么在编辑器中所做的变更就会导致生成的树不符合某个活动工具的预期。例如，这可能意味着工具的查询不再匹配，因而该工具会消失；也可能意味着工具需要从中提取数据的节点已从树中删除。在这两种情况下，HybridSE 都会将违反约束视为不理想的状态，因为这也会使工具进入不一致状态。

因此，我们不会将此变更应用于树。相反，我们会冻结最后一棵树，但立即在编辑器的文本视图中应用文本变更，并将其添加到待处理更改列表中。同时，HybridSE 会要求编辑器提示用户已进入无效状态，例如通过高亮显示或通知。图3展示了我们参考实现中的通知。HybridSE 现在已启动一个事务，后续变更会作为待处理更改变收集起来，并可通过以下三种可能的结果之一解决：

协调 这是理想的情形：用户继续编辑，最终我们会得到一棵不再违反约束的树，无需用户进行特殊干预。用户每次输入后，HybridSE 都会创建一个新的编辑

脚本，从冻结树转换到新的解析结果。这种行为使剪切并粘贴工具或其他会移动工具的操作成为可能：当用户剪切包含约束且该约束要求保留工具的文本范围时，编辑器会进入无效状态。用户粘贴该文本范围后，冻结树（工具位于原始位置）与新树（工具位于新位置）之间的差分启发式算法应能发现，包含工具的子树先被分离，随后又重新附加。从工具的角度来看，工具会被删除的中间状态被跳过了。

应用 用户触发特殊编辑器操作来应用这些变更，而不理会被违反的约束。因此，工具可能会消失。当用户选择使用此操作时，我们假定用户的意图与工具的约束不兼容，而 HybridSE 会优先满足用户的意图。

还原 用户触发特殊编辑器操作，将其变更还原到最后一个有效状态，实质上相当于撤销。通过此操作，待处理更改列表中的变更会被还原，从而使文本视图回到工具约束仍然有效时的状态。

4.1.3 HostEditor 集成

为了将 HybridSE 集成到宿主编辑器中，必须将宿主中执行的所有文本变更报告给列表 5 所示的全局applyChanges 函数。首先，将新变更与任何待处理更改合并，然后应用于当前文本。随后解析这段新文本，并将其与树中最后一个有效的rootNode 进行差分比较。

然后，我们会检查用户是否要求不受工具约束限制地应用这些变更；如果不是，则将编辑脚本传递给validateChanges辅助方法进行验证。该方法会遍历所有约束，检查编辑脚本是否会违反其中任何约束。如果发生违规，则将新变更追加到待处理更改列表中，并回滚编辑脚本。否则，如果编辑脚本未违反任何约束，则可以清空待处理更改列表。

最后，根据验证的结果，我们要么只更新片段（我们将在第4.2节中进一步说明），要么也更新工具，这意味着重新运行其匹配查询，并根据结构的数据提取中任何已更改的结果更新其视图。

4.2 嵌套编辑器

第二个挑战涉及嵌套编辑器，为此我们提供了片段。片段显示一组连续程序节点对应的源代码范围。HybridSE 的差分会维护节点标识，因此可以确定要显示的范围。工具构建者可以定义，已注册工具的全部或部分匹配查询也应在其工具的片段中生效。在片段内部进行匹配，可使工具和片段递归嵌套。由于工具可以嵌入一个允许同一工具进行匹配的片段，因此存在无限递归的风险，HybridSE 为此设置了防护机制。

实现片段的一项复杂之处在于宿主编辑器：设计语言支持系统（如自动补全或错误高亮）的集成，使其在编辑器中仅显示部分范围时仍能正常工作，而这些系统的实现可能默认并不支持这一点。

复杂性的另一个来源是同一源文件的源代码范围以不连续的方式显示：文本导航和文本选择必须考虑屏幕上片段的排列方式以及所显示的文本范围。当文本缓冲区发生变更时，HybridSE 最终会更新片段，如列表 5 所示。

我们调用的函数是 updateFragments，它首先更新每个片段中显示的文本。值得注意的是，片段的范围源自其所显示的节点。因此，当程序树的更新被冻结时，片段的范围可能必须进行偏移，以考虑待处理更改列表中的变更。该函数还会在需要时负责恢复文本选择：如果片段中的偏移导致选择消失或失去同步，HybridSE 会尝试恢复选择，首先在之前选中的片段中进行恢复，然后在视觉上最接近所选片段且包含请求范围的片段中进行恢复。

从工具构建者的角度来看，结构跟踪搭起了从文本到结构的桥梁；而从编辑器用户的角度来看，片段则搭起了从结构回到文本的桥梁：工具构建者声明某些结构应能在工具用户界面中编辑，片段则将这一请求转化为符合用户预期的形式。

由于需要从结构转换为文本，片段必须考虑文本布局中的次级表示，而这在结构化编辑器中无关紧要。这会引出两个问题。

前置与后置空白文本源代码中的表达式通常会包含空白以提高可读性，例如在中缀运算符（如加号）两侧留出空白。由于片段跟踪的是结构，用户扩展表达式时添加的空白可能会被省略，如第 2.2 节所述。为缓解这一问题，我们采用一种默认启发式规则：始终纳入片段表达式右侧的空白；只有当片段表达式左侧的空白不是缩进，且多于一个空格时，才将其纳入。如果片段中显示的结构遵循其他格式约定，尤其是因语言而异时，工具构建者可以覆盖此行为。

缩进 为了解决列表1中所述的缩进不一致问题，片段会计算最大的公共缩进级别（忽略空行），并将该级别以内的缩进替换为一个制表符符号，如图1中缩进行的开头所示。

4.3 结构化访问

第1节提出的第三个挑战涉及读取和编辑结构信息。正如第 3 节所述，实现应提供一个接口，使工具能够访问有关程序树结构的静态信息

。接下来，我们将讨论如何访问动态信息以及编辑结构的方法。

4.3.1 动态信息

为了让动态信息可供访问，宿主平台应提供一种插桩机制，使工具能够在程序结构节点的引用与该节点所取的运行时值之间建立映射。

在我们的参考实现中，这种映射通过重写源代码来实现。如清单 2 第 6 行所示，对于请求运行时值流的节点，我们会重写这些节点，使其将值报告给编辑器。作为一种简单且适用于多种语言、实现成本较低的方法，我们会就地重写表达式，使其连接到 TCP 或 HTTP 服务器。发送的消息包含节点的唯一标识符，以及运行时值的序列化形式。

在我们的参考实现中，在浏览器中运行的 JavaScript，其重写后的表达式形式如下：

```
(e => (
  fetch("https://localhost:3000/watch", {
    method: "POST",
    body: JSON.stringify({ id: $identifier, e }),
    headers: { "Content-Type": "application/json" },
  }), e)
)($expressions)
```

在浏览器环境中，这个表达式可以直接替换插桩后的表达式，无需对文件进行额外修改。通过将插桩代码包装在一个立即求值函数中，我们可以绑定表达式的值，使其只求值一次，并将其同时传递给插桩代码，然后返回该值以继续执行。

实现可以选择将重写结果写入源代码副本，也可以使用 HybridSE 的工具与重写后的表达式进行匹配，并隐藏插桩代码，就像我们在第 3 节中对标记构造所做的那样。如果重写后的表达式写入原始源码，编辑器系统应在代码提交到代码仓库或以其他方式共享之前移除插桩代码。因此，将插桩后的源代码写入副本更可取，但具体选择很可能主要取决于构建链和运行时环境的要求。

4.3.2 结构编辑

在编辑方面，HybridSE 传递给工具构建者的程序结构与底层文本表示之间存在不匹配。获得一个程序结构节点后，工具作者可能会期望能够直接在列表中插入子节点，或将整个子树移动到树中的其他位置。然而，由于这棵树是解析过程的结果，因此所有变更都必须以这样的方式执行：后续对源字符串进行解析时，得到的树结构保持不变。因此，编辑时必须考虑字符串值的转义

、列表元素之间的定界符、二元表达式中的优先级规则和括号，或用户的格式偏好[4]。

例如，假设某个工具想要在 Python 中内联标识符a 。

```
a = 2 + 3
a * 4
```

朴素的文本替换方法会得到：

```
2 + 3* 4
```

由于 Python 定义了隐式优先级规则，使乘法先于加法执行，因此期望的源文本应为：

```
(2 + 3) * 4
```

幸运的是，尽管表层语法往往各不相同，但大多数语言的程序树都采用相似的模式，因此我们可以为编辑提供系统化支持。

原子 就编辑而言，原子是指可以整体替换而不影响其父节点或兄弟节点结构的任何节点。这包括字面值，例如数字或标识符。数组字面量也可以作为原子出现，前提是我们只整体插入它，而不更改其内容。同样，即使字符串中可能包含插值标识符、转义字符或特殊的换行规则，只要将其视为整体，也可以认为它是原子的。字符串中这些嵌套的内容则会以如下所述的列表结构出现。

兄弟节点 树中的兄弟节点出现在使用结构列表的地方。这些列表通常由分隔符字符串隔开。例如，用分号分隔的语句、用花括号界定并以逗号分隔的数组，或用逗号分隔的函数调用参数。在其他列表中，则不需要分隔符：某些可以出现在 S 表达式（嵌套的、由括号界定的列表）中的项具有明确的范围，因此可以直接相邻而不会造成歧义，例如，((b)(c))。

父子结构 这类编程语言中常见的例子是递归表达式层次结构，其中包含中缀、前缀和后缀运算符。编辑这些结构时需要格外谨慎，因为它们受优先级规则约束。

4.3.3 编辑操作

为了处理这些语法结构，HybridSE 定义了四种编辑操作：

node.insert(string,index) 给定一段源代码字符串，将其插入节点的指定索引处，并根据需要添加分隔符。

node.delete() 删除该节点以及所有因此变得多余的分隔符。

node.replaceWith(string) 将该节点替换为源代码字符串，并可能插入括号。

node.wrapWith(string,string) 用指定的前缀和后缀包裹节点，并可选择插入括号。

尽管对结构进行操作却以字符串作为参数，乍看之下似乎不匹配，但我们选择字符串有两个原因：首先，如果参数以常量字符串的形式给出，文本是作者指定要插入的结构的一种简洁方式。其次，如果传入的是现有结构，我们可能仍然更倾向于访问其底层源字符串，以保留用户原有的格式选择。最后，无论哪种情况，都必须将结构序列化为文本。

要执行这些操作，第一步是访问语法中与相关节点对应的定义。HybridSE 要求为每种要用于编辑的语法形式体系实现一个适配器。该适配器必须实现三个函数：(1) 返回给定节点是否位于某个列表中，以及该列表使用的分隔符字符串；(2) 返回要插入列表的第一个位置的字符串索引；(3) 根据文法返回某个节点是否可以用括号包裹。我们的参考实现包含一个用于 Tree-sitter 语法的适配器。

为说明编辑操作，我们将以下简化的语法定义和程序文本作为贯穿全文的示例：

```
函数声明：= 类型 标识符 "(" （类型 ","）* ");" void

```

在兄弟节点中插入根据上述示例，工具作者可以发出命令，将另一个类型作为函数声明的第二项插入，并调用declaration.insert("float",1)。进行插入时，我们首先检查是否存在属于重复规则的子节点，并使用适配器实现找到其分隔符字符串。在 Tree-sitter 适配器中，为此我们首先使用部分解析的简化变体 [2],，尝试将现有程序树节点与语法进行匹配。如果存在多个重复规则，用户可以提供一个函数来选择所需的规则。在本例中，我们找到 （类型","）* 规则和 int 节点。为了识别列表的分隔符，我们使用一组启发式方法，检测 Tree-sitter 语法作者所表达的分隔列表的典型语法结构。其他语法形式体系可能会明确表示定界符，或要求使用不同的启发式方法。下面是三个示例，其中 A 是重复元素，而"S" 是分隔符字符串：

```
至少一个元素，不带尾随分隔符：A ("S" A)* 允
许
尾
列


```

对于不遵循这些约定的程序树模式，例如 Python 的元组（一元素元组显示为(1,)，保留逗号），作者可以在内置启发式方法的基础上添加自己的启发式方法或例外。

我们重复这一过程，直到到达用户请求插入的索引；因此在本例中，由于用户请求在第二个位置插入，我们再搜索一个元素。一旦找到希望在其后插入的现有元素，我们便插入给定的字符串以及此列表的分隔符。如果列表当前为空，也就是找不到任何已位于列表中的现有元素，则使用适配器的函数返回用于在列表开头进行插入的字符串索引。

删除 只有当文法允许其位置为 ϵ（即没有元素），或者它属于重复结构时，才可以删除节点。对于重复结构，我们还希望删除在删除所请求节点后变得多余的任何分隔符。为此，我们采用与插入类似的方法：首先检测列表是否包含分隔符。然后检查该节点的相邻兄弟节点中是否有这样的分隔符，若有，则一并删除。

替换 如果用户请求替换一个节点，我们会将其理解为：请求把给定的字符串逻辑上放置在树中与待替换节点相同的位置。因此，我们可能需要确保设置了括号。为此，我们首先询问文法适配器，查看是否有支持括号的文法规则。如果有，我们就插入用户要求的、不带括号的文本，重新解析，并尝试访问一个范围与插入字符串的确切范围相同的程序树节点。如果找不到这样的节点，说明结合性导致我们插入的部分被分配到了不同的子树中，因此我们会用括号重复替换操作。

5 参考实现

为了证明我们提出的界面在技术上可行，我们为 CodeMirror v6 编辑器实现了该界面，并为 Lively 42 和 Squeak/Smalltalk [23] 起草了两种实现方案，以识别技术难点。该实现可作为制品访问 [6]。

我们的 HybridSE 参考实现基于 CodeMirror v6 [29], ，这是一款流行的浏览器编辑器库。CodeMirror v6 采用函数式库设计，并注重可扩展性。因此，它是一个很好的起点。作为编辑器库，它并不是完整的集成开发环境，例如，默认不集成自动补全等服务，因此只显示文件的一部分时不会遇到挑战。

作为第二款编辑器，我们选择了 Lively 4 的默认编辑器。Lively 4 是一款基于浏览器的集成开发环境。它的编辑器组件也基于 CodeMirror，但使用的是第 5 版，该版本提供的是命令式API。由于 CodeMirror v6 和 Lively 都基于浏览器，我们能够复用 HybridSE 的核心实现，例如解析和差分算法，只需重新实现便于与编辑器集成的功能，这也展示了它作为独立、可复用 API 的潜在用途。Lively 4 集成了多种 JavaScript 语言服务。

最后，我们将 HybridSE 扩展到了基于 Morphic 实现的 Squeak/Smalltalk 默认编辑器中 [28] 。为此，我们需要为 Smalltalk 重新实现 HybridSE 的核心，并在 Squeak 的核心方法中添加扩展点，因为其默认编辑器在设计上并不支持针对编辑事件进行扩展。不过，Squeak 的文本编辑器组件本身很容易配置，可为程序的各个子部分提供预期的语言服务。

[2]https://github.com/LivelyKernel/lively4-core，最后访问时间：2025-09-27。

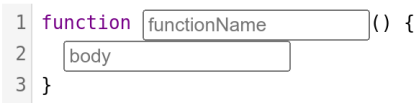


图4 用于 JavaScript 表达式或标识符的占位文本字段。用户开始在文本字段中输入时，字段会自动替换为输入的文本。

除了参考实现和两个实现构想之外，我们还开始评估其他流行编辑器是否具备这些能力。此前已有针对 JetBrains 集成开发环境 [24],的工具展示了与源代码的类似紧密集成方式 [32]。其用于创建扩展的 API 足够强大，能够满足全部三项标准。

Visual Studio Code 的 [34]扩展 API 仅允许有限地访问编辑器组件。其内部 API 目前不支持嵌入任意大小、仅占据一行部分空间的组件。相反，嵌入式组件需要占据整个水平空间，或以弹出窗口的形式显示。因此，要集成 HybridSE，就需要变更 Visual Studio Code 编辑器组件的实现。

在 HybridSE 的参考实现中，我们使用 Tree-sitter [30] 进行解析。由于 Tree-sitter 仅以可执行文件的形式发布解析器，因此我们会根据编辑需要，从其开源解析器代码库中访问语法定义（第4.3.3节）。虽然 HybridSE 可与任何解析器配合使用，而不仅限于 Tree-sitter，但从不同类型的语法库中推导分隔符或优先级的启发式方法必须针对每种新类型进行调整。

6 案例研究

为了评估使用 HybridSE 可以创建的工具范围，我们选择了小型工具（它们以不起眼的方式添加到文本编辑器中），也选择了大型重构工具（它们旨在将基于文件的编辑转变为更具可视化的界面）。

6.1 在程序中插入占位文本

工具有时需要插入仍需用户输入的表达式。为此，HybridSE 建议定义一种小型工具，在代码尚未补全的位置显示占位符，如图 4 所示。

该工具会搜索标记字符串 `__VI_占位符_{label}`，例如搜索 `__VI_占位符_body`，以创建一个标签为 `body` 的占位文本字段。用户开始在文本字段中输入内容时，该字段会被新输入的内容替换。这是一种便捷方式，工具构建者可以借此告知其他工具的用户哪些位置仍需输入，同时仍能生成不缺少标识符的有效程序树。

占位符工具的实现需要 36 行代码，如代码清单 3 所示。该工具定义了一种特殊约束，不仅要求节点保持存在，还要求节点内容保持不变，因为该工具依赖于节点内容的精确匹配。

```
const db = pool.connect();
db.query(sql`SELECT * FROM `events` WHERE id = 123`);
```

图 5 嵌入在 JavaScript 编辑器中的 SQL 编辑器。嵌套的 SQL 定义在由反引号界定的字符串中，且其中本身也包含反引号，但无需手动转义，因为 HybridSE 会以正确的格式转换任何变更。

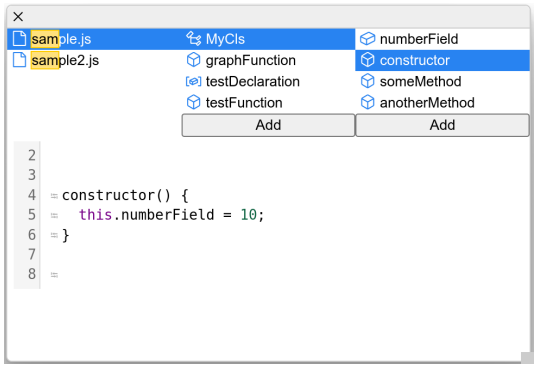


图 6 用于 JavaScript 类的浏览器工具。

6.2 组合语言

库通常允许将其他编程语言的源文本指定为字符串。一个常见的例子是 SQL，它以字符串的形式传递给数据库。在宿主语言的字符串中存储第二种语言时，用户必须遵循转义规则，这可能会使表达式变得更加复杂。一般而言，不进行转义就无法组合两种文本语言。这种能力是使用结构而非解析的编辑器的一项优势 [13, 44]。

由于 HybridSE 向工具提供了结构化接口，因此很好地支持这种类型的组合。我们定义一个工具，用于匹配包含第二种语言的嵌入字符串。提取字符串内容时，我们会移除转义，并在一个配置为嵌入语言的新片段中显示未转义的字符串。在图 5 中，我们展示了一个嵌入 SQL 编辑器的 JavaScript 编辑器。请注意，反引号既用作 JavaScript 中的字符串分隔符，也出现在 SQL 中。用户输入字符时，我们会检查该字符是否需要转义；如果需要，就直接在宿主编程语言的字符串中插入转义后的字符。

由于工具可能经常需要对字符串进行反转义，我们的参考实现为 JavaScript 提供了内置功能。借助该内置功能，如清单 4 所示，工具的定义仅包含 17 行代码。内置功能本身包含 50 行代码，因为它还支持在转义和未转义版本之间转换光标索引。

6.3 JavaScript 结构化代码浏览器

受 Smalltalk 的代码浏览器 [23] 以及 CodeBubble 将 Java 文件重新映射为按定义组织的结构这一做法的启发，我们使用 HybridSE 构建了一个类似浏览器的界面

。图 6 展示了在浏览器中打开的 JavaScript 文件截图。浏览器界面将源文件拆分为可单独编辑的声明。这带来了一系列 HybridSE 能够帮助工具构建者解决的挑战。

为了能够在浏览器中打开编程语言文件，我们需要一种映射，用于定义相应语言中的声明如何嵌套。可选的是，该映射还可以为不同类型的声明定义图标和标签。对于 JavaScript，我们既为顶层节点（即程序节点的每个直接子节点）定义了图标映射，也定义了从 JavaScript 类中提取嵌套声明的映射。为了使顶层节点列表保持可用，我们会合并某些连续节点类型，例如导入语句。

用户打开项目文件夹后，浏览器界面便会出现。左栏列出所有项目文件。中栏显示所选文件的顶层语句。右栏显示所选顶层语句的所有成员。下方的面板包含一个文本编辑器，仅显示选定的程序节点。

浏览器界面要求能够单独编辑各个声明。间歇性的变更可能会导致源树中前置或后续的声明受损。例如，假设有一个包含两个顶层语句的 JavaScript 程序：

```
1 var a = 5
2 b
```

如果用户在 5 后面输入加号，程序就会被解析为一个形如 vara = 5 + b 的单一语句，尽管它显示在多行上。因此，当用户输入加号时，我们在浏览器编辑器中显示的语句会突然将第二条语句也纳入其中。为防止这种情况，我们定义一项约束，确保所选节点以及前后相邻的节点都保持为顶层语句，从而避免用户构思变更时，后续或前面的语句被纳入其中。

当用户在顶层列表或成员列表中点击添加按钮时，我们会调用第4.3.3节所述的插入方法，并传入第6.1节所述形式的占位符。插入方法会返回新创建的符号，使我们能够立即在编辑视图中选中它。

浏览器的实现包括用 200 行代码定义其用户界面，其中还包含用于选择的状态管理。另外，约 160 行以查询语句为主的代码实现了对顶层语句及其嵌套成员的匹配。其中还包括合并连续的顶层项目，避免将它们作为独立项目处理，例如多行注释和文件导入列表。

6.4 “Livelits”的用户界面

我们的最后一个案例研究如图 7 所示，实现了 Livelits [37], 的用户界面组件，满足其对持久性、组合性、参数化以及部分活性的要求。例如，滑块使用 [“滑块”, 0, 255, 1, 73][1]形式，实现工具的持久性和参数化（指定最小值、

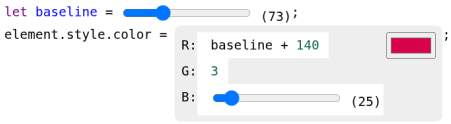


图 7 重新实现 Livelits [37] 的滑块和颜色工具。

最大值和步长）。此外，颜色选择器会访问红色、绿色和蓝色的运行时值，以实现活性。组合性则通过 HybridSE 的片段实现。由于我们的示例是用 JavaScript 实现的，因此未涉及类型要求。我们的工作也未涉及去中心化可扩展性要求——不过，我们的参考实现确实会将工具定义与使用这些工具的库打包在一起。

7 讨论与未来工作

接下来，我们将讨论所提出方法的局限性以及可能的未来工作。

编辑器集成的可行性 如第 5 节所述，即使采用所提出的方法，如果宿主编辑器没有为集成文本以外的可视元素做好准备，仍可能需要投入大量工程工作。不过，由于 CodeMirror 等开源文本编辑器能够提供兼容的接口，因此这似乎不存在概念上的限制。性能方面可能存在一些顾虑：对大型文件运行匹配查询并实例化大量工具，可能会降低整体性能。值得注意的是，工具可能会改变行高。这会使一种常见的优化变得复杂：当所有行的高度相同时，编辑器可以根据给定的滚动位置计算出精确的行范围，从而跳过不必要的文本布局操作。

对静态和动态信息的结构化访问我们认为，HybridSE 提供的静态和动态信息访问接口展现了这样一种接口的价值：通过结构节点引用来便利地访问信息。目前，静态信息仅限于程序结构的语法层。对于某些工具而言，也可能需要访问语义信息，例如对变量的引用。为此，可以建立节点标识与语言服务器所提供功能之间的映射 [33]。

在动态信息方面，通过源代码重写展示的方法，为结构引用与观测到的运行时值之间建立映射提供了一种非常简单的方式。若要进一步扩展这一界面，应考虑信息来源，例如测试运行、示例执行或实际运行的程序。此外，若能提供更详细的执行信息，界面将更为完善，例如执行语句时的堆栈跟踪；理想情况下，编辑器中的栈帧可以直接映射到程序结构引用。

源代码中的工具注解 我们展示的许多工具都利用程序树的结构来实现功能，有时还会主动修改程序树以满足自身需求，例如 ["__监视器", expr][1] 标记。在某些情况下，这些标记被视为永久内容，应纳入版本控制；例如，在第6.4节中，某个数字应显示为滑块的注解。在其他情况下，它们也可以是用于跟踪的临时注解，如第4.3.1节所述。这种方法使工具能够在不扩展语言语法的情况下与源代码集成；否则，文件将无法与现有的语言支持工具兼容。此外，它还允许将工具信息保存在源文件中，以便文本版本控制系统能够处理这些信息。

这与在注释中或单独文件里添加注解形成对比——这两种方法可能都不依赖我们的重写方法所要求的那个令人意外的 [1] ，但对于不支持 HybridSE 的编辑器中的变更，它们可能更容易出错，例如同事可能会使用这类编辑器。支持 HybridSE 且能识别注释或单独文件中注解的编辑器，可以确保更新这些注解；但用户可能会忘记手动更新，例如在网页界面中进行合并时。

与其他形式的注解相比，表达式重写方法受到语言语法的约束：例如，第 6.1 节占位符案例研究中的方法，不允许工具创建下拉菜单，以便选择性地添加关键字，例如 Java 中的 public ，因为必须更改该关键字，才能将其识别为占位符。

与将结构而非文本存储起来的结构化编辑器相比，例如在 MPS 中实现的编辑器 [10], ，我们的方法给工具构建者带来了更多挑战。无需手动指定约束，对程序节点的注解会自动持久化，而且从工具的结构化视图切换到用户的文本视图时也不会出现不匹配。因此，我们提出的混合方法主要为编辑器用户创造了价值，同时也使工具构建者能够利用现有工具可能更庞大的生态系统，因为这些工具需要文本。

易用性 如上所述，我们的方法允许用户继续使用文本编辑器，同时也能受益于以代码为中心的工具。我们的参考实现尚未投入大量精力来优化用户体验，未来的工作应开展用户研究，以确认所提出的交互是否能够无误地运行。总体而言，由于交互方式沿用了底层文本编辑器中用户熟悉的操作，因此在非正式测试中，除了了解特定工具的功能外，用户无需额外说明即可开始使用该编辑器。

通过亲自使用参考实现，我们注意到它处理约束的方式存在两个主要问题：我们显示的弹出窗口显得相当突兀。采用更含蓄的设计可以显示更小的提示。此外，与其使用我们的方法所提出的通用函数，不如仅使用一组内置约束，这样 HybridSE 或许能够更准确地推断出具体是什么改动导致约束失效，从而向用户提供更好的反馈；尤其是在多个工具的约束均失效，导致程序树“冻结”时。

其次，HybridSE 缺少一种启发式方法，无法识别用户是否意图删除某个工具：当用户选中一段较大的文本范围，其中包含一个工具，并按下退格键时，某个要求保留在程序树中的工具所设定的约束就会被违反。区分退格键和剪切操作，或许能让 HybridSE 更好地推断用户意图。

8 结论

创建以代码为中心的工具面临多项挑战，因为文本形式只能间歇性地产生工具可访问的有效树结构。结构化编辑器解决了这一挑战，但其用户界面对于从文本编辑器转来的用户来说可能并不熟悉，而且还需要重新创建编程语言生态系统中的很大一部分，而该生态系统以文本为基础。

我们提出了混合结构化编辑（HybridSE），它为工具提供完全结构化的接口，同时为用户提供完全文本化的界面。作为连接这两个世界的桥梁，HybridSE 允许工具有选择地指定对结构的约束，并在程序树的无效状态之间使用树差异比较和事务，让工具能够将程序树视为始终可用。通过案例研究，我们展示了 HybridSE 适用于创建以代码为中心的工具：与文本缓冲区集成以及访问结构中的信息所带来的复杂性，都转移到了 HybridSE 上。因此，混合结构化编辑为文本编辑器提供了一种工具扩展机制，帮助用户理解源代码与工具所推导出的信息之间的联系。

致谢 我们感谢匿名审稿人提供的宝贵反馈。本研究由 SAP 和 HPI–MIT“面向可持续性的设计”研究项目资助。[3]

A 代码清单

清单 3 第 6.1 节所述占位符工具的完整声明。

```
1 const placeholderTool  = { 2 查询: 节点 =>{ 3 // 为标识符映射特
定
识
名
```

[3] https://hpi.de/en/research/cooperations-partners/research-program-designing-for-sustainability.html，最后访问时间：2025-02-04。

```
10 }[节点.language]; 11 if (node.type  == identifierType && node.text.startsWith("__VI_
PL
变
 =>
re
标
28
int
re
```

列表 4 第 6.2 节所述 sql 工具的完整声明。

```
1 const sqlTool  = { 2 query: node =>
{ 3
nu
els
tex
va
e.d
{sq
```

列表 5 向 HybridSE 通知变更的函数。

```
1 defapplyChanges(变更, force应用  = false):2 oldText = hybridSE.text3 hybridSE.text =
a
p
```

```
5 if not force应用 and not 验证Changes(editScript, 变更): 6 hybridSE.text =oldText 7
up
Ch
va
15
tru
```

B 源代码引用

以下文件引用指向制品 [6] （下文使用的术语“vitrail”是我们为 HybridSE 参考实现所选的名称）。

- 第4.1.1节，差分：core/diff.js第4.1.2节，事务：部分内容位于applyChanges 函数
- 中，位于 vitrail/vitrail.ts:532；CodeMirror 的待定变更通知实现位于 vitrail/codemirror6.ts:449。第4.2节，嵌套编辑器： vitrail/pane.ts；空白处理位于
- core/query.js:218；缩进处理：vitrail/tools/whitespace.ts第4.3.1节，动态信息：vitrail/tools/watch.ts第4.3.3节，编辑操作：core/model.js:821 以及 Tree-sitter 的实现 core/tree-sitter.js:337
-

参考文献

[1] Leif Andersen、Michael Ballantyne 和 Matthias Felleisen。《为文本代码添加交互式可视化语法》。载于：制品：为文本代码添加交互式可视化语法 4.OOPSLA（2020年11月），222:1–222:28。doi：10.1145/3428290。[2] Tom Beckmann、Patrick Rein、Toni Mattis 和 Robert Hirschfeld。《结构化编辑器的部分解析》。载于：第*15*届 *ACM SIGPLAN* 软件语言工程国际会议论文集。SLE 2022。美国纽约州纽约市：计算机协会，2022年12月，页码 110–120。ISBN：978-1-4503-9919-7。doi：10.1145/3567512.3567522。[3] Tom Beckmann、Patrick Rein、Stefan Ramson、Joana Bergsiek 和 Robert Hirschfeld。《人人皆可用的结构化编辑：从语法推导出易用的结构化编辑器》。载于：*2023*年计算系统人为因素*CHI*会议论文集

计算系统。德国汉堡：ACM，2023年4月，页码 1–16。ISBN：978-1-4503-9421-5。doi：10.1145/3544548.3580785。[4] Tom Beckmann、Jan Reppien、Jens Lincke 和 Robert Hirschfeld。《支持在文本源代码中构建领域特定表示》。载于：第*3*届 *ACM SIGPLAN* 编程抽象、交互式符号、工具与环境国际研讨会论文集。PAINT '24。美国纽约州纽约市：计算机协会，2024年10月，页码 17–28。ISBN：979-8-4007-1212-8。doi：10.1145/3689488.3689990。[5] Tom Beckmann、Daniel Stachnik、Jens Lincke 和 Robert Hirschfeld。《可视化替换：结构化编辑器中的跨语言领域特定表示》。载于：第*2*届 *ACM SIGPLAN* 编程抽象、交互式符号、工具与环境国际研讨会论文集。PAINT 2023。美国纽约州纽约市：计算机协会，2023年10月，页码 25–35。ISBN：979-8-4007-0399-7。doi：10.1145/3623504.3623569。[6] Tom Beckmann、Christoph Thiede、Jens Lincke 和 Robert Hirschfeld。《"混合结构化编辑：面向工具的结构，面向用户的文本"的已接收制品。2026年2月。doi：10.5281/zenodo.18455580。[7] Thorsten Berger、Markus Völter、Hans Peter Jensen、Taweesap Dangprasert 和 Janet Siegmund。《投影式编辑的效率：一项受控实验》。载于：*2016*年第*24*届 *ACM SIGSOFT* 软件工程基础国际研讨会论文集。FSE 2016。美国纽约州纽约市：计算机协会，2016年11月，页码 763–774。ISBN：978-1-4503-4218-6。doi：10.1145/2950290.2950315。[8] Marat Boshernitsan。*Harmonia*：用于构建基于语言的编程工具的灵活框架。技术报告 UCB/CSD-01-1149。2001年6月。网址：http://www2.eecs.berkeley.edu/Pubs/TechRpts/2001/5819.html。[9] Andrew Bragdon、Robert Zeleznik、Steven P. Reiss、Suman Karumuri、William Cheung、Joshua Kaplan、Christopher Coleman、Ferdi Adeputra 和 Joseph J. LaViola。《Code Bubbles：用于代码理解与维护的基于工作集的界面》。载于：*SIGCHI*计算机系统人为因素会议论文集。美国佐治亚州亚特兰大：ACM，2010年4月，页码 2503–2512。doi：10.1145/1753326.1753706。[10] Fabien Campagne 和 Fabien Campagne。*MPS*语言工作台，第*1*卷。第1版。美国南卡罗来纳州北查尔斯顿：CreateSpace独立出版平台，2014年5月。ISBN：978-1-4973-7865-0。[11] Andrei Chiş、Oscar Nierstrasz 和 Tudor Gîrba。《迈向可塑造的开发工具》。载于：第*6*届编程语言与工具的评估和可用性研讨会论文集。美国宾夕法尼亚州匹兹堡：ACM，2015年10月，页码 25–26。ISBN：978-1-4503-3907-0。doi：10.1145/2846680.2846684。

[12] Lukas Diekmann 和 Laurence Tratt。《别慌！为 LR 解析器提供更好、更少的语法错误》。载于：第*34*届欧洲面向对象编程会议（*ECOOP 2020*）。Robert Hirschfeld 和 Tobias Pape 编。第166卷。《莱布尼茨国际信息学会议论文集》（LIPIcs）。ISSN：1868-8969。德国达格施图尔：Schloss Dagstuhl – Leibniz-Zentrum für Informatik，2020年，6:1–6:32。ISBN：978-3-95977-154-2。DOI：10.4230/LIPIcs.ECOOP.2020.6。[13] Lukas Diekmann 和 Laurence Tratt。《Eco：语言组合编辑器》。载于：软件语言工程。Benoît Combemale、David J. Pearce、Olivier Barais 和 Jurgen J. Vinju 编。第8706卷。丛书名：计算机科学讲义。沙姆：施普林格国际出版，2014年，第82–101页。ISBN：978-3-319-11245-9。DOI：10.1007/978-3-319-11245-9_5。[14] Patrick Dubroy。*Moonchild*。未来编程研讨会（FP），SPLASH 2014。2014年。网址：http://www.future-programming.org/2014/program.html。[15] Patrick Dubroy 和 Alessandro Warth。《增量 Packrat 解析》。载于：第*10*届 *ACM SIGPLAN* 软件语言工程国际会议论文集。SLE 2017。美国纽约州纽约市：计算机协会，2017年10月，第14–25页。ISBN：978-1-4503-5525-4。DOI：10.1145/3136014.3136022。[16] Jonathan Edwards 和 Tomas Petricek。《交互与抽象：受控复制粘贴》。载于：第*1*届 *ACM SIGPLAN* 编程抽象、交互式符号、工具与环境国际研讨会论文集。PAINT 2022。美国纽约州纽约市：计算机协会，2022年12月，第11–19页。ISBN：978-1-4503-9910-4。DOI：10.1145/3563836.3568723。[17] Jonathan Edwards 和 Tomas Petricek。《借助结构化编辑实现基于图像的类型化编程》。发表于人类类型与推理助理研讨会（HATRA）。在线地址：http://tomasp.net/academic/papers/typed-image。2021年。[18] Sebastian Erdweg、Tijs van der Storm、Markus Völter、Laurence Tratt、Remi Bosman、William R. Cook、Albert Gerritsen、Angelo Hulshout、Steven Kelly、Alex Loh、Gabriël Konat、Pedro J. Molina、Martin Palatnik、Risto Pohjonen、Eugen Schindler、Klemens Schindler、Riccardo Solmi、Vlad Vergu、Eelco Visser、Kevin van der Vlist、Guido Wachsmuth 和 Jimi van der Woning。《语言工作台的评估与比较：现有结果及未来基准》。载于：计算机语言、系统与结构。第6届和第7届软件语言工程国际会议（SLE 2013和SLE 2014）特刊，第44卷（2015年12月），第24–47页。ISSN：1477-8424。DOI：10.1016/j.cl.2015.08.007。[19] Sebastian Erdweg、Tamás Szabó 和 André Pacak。《简洁、类型安全且高效的结构化差分》。载于：第*42*届 *ACM SIGPLAN* 编程语言设计与实现国际会议论文集。加拿大线上会场：ACM，2021年6月，第406–419页。ISBN：978-1-4503-8391-2。DOI：10.1145/3453483.3454052。

[20] Jean-Rémy Falleri、Floréal Morandat、Xavier Blanc、Matias Martinez 和 Martin Monperrus。《细粒度且准确的源代码差异比较》。载于：第*29*届 *ACM/IEEE* 国际自动化软件工程会议论文集。ASE '14。美国纽约州纽约市：计算机协会，2014年9月，第313–324页。ISBN：978-1-4503-3013-8。DOI：10.1145/2642937.2642982。[21] Camille Gobert 和 Michel Beaudouin-Lafon。《Lorgnette：创建可塑代码投影》。载于：第*36*届 *ACM* 用户界面软件与技术研讨会论文集。美国加利福尼亚州旧金山：ACM，2023年10月，第1–16页。ISBN：979-8-4007-0132-0。DOI：10.1145/3586183.3606817。[22] Joshua Horowitz 和 Jeffrey Heer。《Engraft：用于实时、丰富且可组合编程的 API》。载于：第*36*届 *ACM* 用户界面软件与技术研讨会论文集。美国加利福尼亚州旧金山：ACM，2023年10月，第1–18页。ISBN：979-8-4007-0132-0。DOI：10.1145/3586183.3606733。[23] Dan Ingalls、Ted Kaehler、John Maloney、Scott Wallace 和 Alan Kay。《回到未来：Squeak 的故事——一种用自身编写的实用 Smalltalk》。载于：第*12*届 *ACM SIGPLAN* 面向对象编程、系统、语言与应用会议论文集。OOPSLA '97。美国纽约州纽约市：计算机协会，1997年10月，第318–326页。ISBN：978-0-89791-908-1。DOI：10.1145/263698.263754。[24] JetBrains。*IntelliJ IDEA*。2001年。网址：https://www.jetbrains.com/idea/（访问于2025-08-15）。[25] John Lenz 和 Nick Fitzgerald。源映射修订版*3*提案。2011年。网址：https://sourcemaps.info/spec.html （访问于2025-09-25）。[26] Sven Keidel、Wulf Pfeiffer 和 Sebastian Erdweg。《IDE可移植性问题及其在 Monto 中的解决方案》。载于：*2016*年 *ACM SIGPLAN* 软件语言工程国际会议论文集。SLE 2016。美国纽约州纽约市：计算机协会，2016年10月，第152–162页。ISBN：978-1-4503-4447-0。DOI：10.1145/2997364.2997368。[27] Amy J. Ko 和 Brad A. Myers。《Barista：为代码编辑器启用新工具、交互技术和视图的实现框架》。载于：*SIGCHI*计算机系统人为因素会议论文集。加拿大魁北克省蒙特利尔：ACM，2006年4月，第387–396页。ISBN：978-1-59593- 372-0。DOI：10.1145/1124772.1124831。[28] John H. Maloney 和 Randall B. Smith。《Morphic用户界面构建环境中的直接性与活性》。载于：第*8*届 *ACM* 用户界面与软件技术年度研讨会论文集。UIST '95。美国纽约州纽约市：计算机协会，1995年12月，第21–28页。ISBN：978-0-89791-709-4。DOI：10.1145/215585.215636。[29] Marijn Haverbeke。*CodeMirror*。2007年。网址：http://codemirror.net/（访问于2025-09-21）。

[30] Max Brunsfeld。*Tree-sitter*。2020年。网址：https://tree-sitter.github.io/tree-sitter/（访问于2025年8月15日）。[31] Mauricio Verano Merino、Jur Bartels、Mark van den Brand、Tijs van der Storm 和 Eugen Schindler。《投影式文本语言》。载于：实践中的领域专用语言：使用 *JetBrains MPS*。Antonio Bucchiarone、Antonio Cicchetti、Federico Ciccozzi 和 Alfonso Pierantonio 编。沙姆：施普林格国际出版，2021年，197–225页。ISBN：978-3-030-73758-0。DOI：10.1007/978-3-030-73758-0_7。[32] Paul Methfessel、Tom Beckmann、Patrick Rein、Stefan Ramson 和 Robert Hirschfeld。《MµSE：通过示例支持软硬件交互探索》。载于：计算机系统人为因素*CHI*会议论文集。美国夏威夷州檀香山：ACM，2024年5月，1–16页。ISBN：979-8-4007-0330-0。DOI：10.1145/3613904.3642186。[33] 微软。语言服务器协议。网址：https://microsoft.github.io/language-server-protocol/（访问于2025年8月22日）。[34] 微软。*Visual Studio Code*。2010年。网址：https://code.visualstudio.com/（访问于2025年8月15日）。[35] David Moon、Andrew Blinn 和 Cyrus Omar。《带义务的渐进式结构编辑》。载于：*2023*年*IEEE*可视化语言与以人为中心的计算研讨会（*VL/HCC*）。美国华盛顿特区：IEEE，2023年10月，71–81页。DOI：10.1109/vl-hcc57772.2023.00016。[36] Fabio Niephaus、Eva Krebs、Christian Flach、Jens Lincke 和 Robert Hirschfeld。《PolyJuS：基于Squeak/Smalltalk、面向GraalVM的多语言笔记本系统》。载于：第*3*届编程艺术、科学与工程国际会议附属会议论文集。意大利热那亚：ACM，2019年4月，1–6页。ISBN：978-1-4503-6257-3。DOI：10.1145/3328433.3328434。[37] Cyrus Omar、David Moon、Andrew Blinn、Ian Voysey、Nick Collins 和 Ravi Chugh。《使用实时图形用户界面填充类型化空洞》。载于：第 *42* 届 *ACM SIGPLAN*编程语言设计与实现国际会议论文集。加拿大线上会场：ACM，2021年6月，511–525页。ISBN：978-1-4503-8391-2。DOI：10.1145/3453483.3454059。[38] Jupyter项目。*Jupyter*项目。2015年。网址：https://jupyter.org（访问于2025年8月15日）。[39] David Rauch、Patrick Rein、Stefan Ramson、Jens Lincke 和 Robert Hirschfeld。《巴比伦式编程：将实时示例集成到通用源代码中的设计与实现》。载于：《编程的艺术、科学与工程》3.3（2019年2月），第9页。ISSN：2473-7321。DOI：10.22152/programming-journal.org/2019/3/9。

[40] Charles Simonyi、Magnus Christerson 和 Shane Clifford。《意向式软件》。载于：第*21*届*ACM SIGPLAN*面向对象编程系统、语言与应用年会论文集。OOPSLA '06。美国纽约州纽约市：计算机协会，2006年10月，451–464页。ISBN：978-1-59593-348-5。DOI：10.1145/1167473.1167511。[41] Tamás Szabó、Markus Voelter、Bernd Kolb、Daniel Ratiu 和 Bernhard Schaetz。《mbeddr：面向嵌入式软件开发的可扩展语言》。载于：*AdaLett.* 34.3（2014年10月），13–16页。ISSN：1094-3641。DOI：10.1145/2692956.2663186。[42] Marcel Taeumel、Bastian Steinert 和 Robert Hirschfeld。《VIVIDE编程环境：将运行时信息与程序员的系统知识相结合》。载于：*ACM*国际编程与软件新理念、新范式及反思研讨会论文集。美国亚利桑那州图森：ACM，2012年10月，117–126页。ISBN：978-1-4503-1562-3。DOI：10.1145/2384592.2384604。[43] David Ungar、Henry Lieberman 和 Christopher Fry。《调试与即时体验》。载于：*Commun. ACM* 40.4（1997年4月），38–43页。ISSN：0001-0782。DOI：10.1145/248448.248457。[44] Markus Voelter、Janet Siegmund、Thorsten Berger 和 Bernd Kolb。《迈向用户友好的投影式编辑器》。载于：软件语言工程。Benoît Combemale、David J. Pearce、Olivier Barais 和 Jurgen J. Vinju 编。沙姆：施普林格国际出版，2014年，41–61页。ISBN：978-3-319-11245-9。DOI：10.1007/978-3-319-11245-9_3。[45] Markus Voelter、Tamás Szabó、Sascha Lisson、Bernd Kolb、Sebastian Erdweg 和 Thorsten Berger。《使用语法单元高效开发一致的投影式编辑器》。载于：*2016*年*ACM SIGPLAN*软件语言工程国际会议论文集。荷兰阿姆斯特丹：ACM，2016年10月，28–40页。ISBN：978-1-4503-4447-0。DOI：10.1145/2997364.2997365。[46] Tim A. Wagner 和 Susan L. Graham。《高效且灵活的增量解析》。载于：*ACM Trans. Program. Lang. Syst.*20.5（1998年9月），980–1013页。ISSN：0164-0925。DOI：10.1145/293677.293678。

作者简介

Tom Beckmann 是波茨坦大学哈索·普拉特纳研究所软件架构组的成员。他致力于通用语言的结构化编辑，以更好地支持工具集成。他目前的研究兴趣包括编程工具设计，以及面向编程的编辑与输入方法。请通过 tom.beckmann@hpi.uni-potsdam.de 联系 Tom。

https://orcid.org/0000-0003-0015-1717

Christoph Thiede 是波茨坦大学哈索·普拉特纳研究所软件架构组的成员。他目前的研究兴趣包括探索式编程工作流、工具设计，以及分析和操纵程序执行的技术。请通过 christoph.thiede@hpi.uni-potsdam.de联系他。

https://orcid.org/0000-0002-7442-8216

Jens Lincke 是哈索·普拉特纳研究所软件架构组的成员，他对实时与探索式编程（Lively Kernel）感兴趣。他因一篇关于在协作式自我支持开发环境中演进工具的论文而获得博士学位。可通过jens.lincke@hpi.uni-potsdam.de与他联系。

https://开放研究者与贡献者身份识别码.org/0000-0002-3828-7778

Robert Hirschfeld 领导波茨坦大学哈索·普拉特纳研究所的软件架构组。他的研究兴趣包括动态编程语言、开发工具和运行时环境，旨在使实时、探索式编程更易于上手。可通过 robert.hirschfeld@hpi.uni-potsdam.de联系 Robert。

https://orcid.org/0000-0002-4249-6003