IDE Support for Lexical Effects and Handlers

Bachelor‘s Thesis Presentation

Tim Neumann

Student at Faculty of Science

Eberhard Karls Universität Tübingen

February 16, 2022

Overview

1. Effects and Handlers
2. Research on Language Support in IDEs
3. Proposals on IDE Support for Effects and Handlers
4. Conclusion
5. Bibliography

Effects and Handlers

Effects and Handlers

Published: February 2003

Algebraic Operations and Generic Effects

Gordon Plotkin & John Power

Applied Categorical Structures **11**, 69–94 (2003) | Cite this article

411 Accesses | **138** Citations | Metrics

Abstract

Given a complete and cocomplete symmetric monoidal closed category V and a symmetric monoidal V-category C with cotensors and a strong V-monad T on C, we investigate axioms under which an $Ob\ C$-indexed family of operations of the form $\alpha_x:(Tx)^v \to (Tx)^w$ provides semantics for algebraic operations on the computational λ-calculus. We recall a definition for which we have elsewhere given adequacy results, and we show that an enrichment of it is equivalent to a range of other possible natural definitions of algebraic operation. In particular, we define the notion of generic effect and show that to give a generic effect is equivalent to giving an algebraic operation. We further show how the usual monadic semantics of the computational λ-calculus extends uniformly to incorporate generic effects. We outline examples and non-examples and we show that our definition also enriches one for call-by-name languages with effects.

[Plotkin and Power, 2003]

Effects and Handlers

Describing an effect

```
effect Logging {
  def logResult[A](s: A) : A
  def logMessage(s: String) : String
}
```

Example code will be in the Effekt language [Brachthäuser et al., 2020]

Effects and Handlers

Utilising effect operations

```
def add(a: Int, b: Int) : Int / { Logging } =
{
    val result = a + b;
    logMessage(result.show);
    result;
}
```

Effects and Handlers

Effect handlers

```
def main() : Int / { Logging } = {
  add(23, 42)
}
```

Effects and Handlers

Effect handlers

```
def main() : Int / { Logging } = {
  add(23, 42)
}
```

Missing implementation

```
REPL> main()
[error] Main cannot have user defined effects, but includes
effects: Logging
```

Effects and Handlers

Effect handlers

```
def main() : Int / { Logging } = {
  add(23, 42)
}
```

How to implement the functionality of *Logging*?

Effects and Handlers

European Symposium on Programming
ESOP 2009: Programming Languages and Systems pp 80-94 | Cite as

Handlers of Algebraic Effects

Authors Authors and affiliations

Gordon Plotkin, Matija Pretnar

Conference paper 73 Citations 1.4k Downloads

Part of the Lecture Notes in Computer Science book series (LNCS, volume 5502)

Abstract

We present an algebraic treatment of exception handlers and, more generally, introduce handlers for other computational effects representable by an algebraic theory. These include nondeterminism, interactive input/output, concurrency, state, time, and their combinations; in all cases the computation monad is the free-model monad of the theory. Each such handler corresponds to a model of the theory for the effects at hand. The handling construct, which applies a handler to a computation, is based on the one introduced by Benton and Kennedy, and is interpreted using the homomorphism induced by the universal property of the free model. This general construct can be used to describe previously unrelated concepts from both theory and practice.

[Plotkin and Pretnar, 2009]

Effects and Handlers

Exception handlers

```
try {
    something()
} catch(Exception e) {
    System.out.println("Uh oh...")
}
```

Effects and Handlers

Effect handlers

```
def main() : Int / { Logging } = {
    add(23, 42)



}
```

Effects and Handlers

Effect handlers

```
def main() : Int / { Logging } = {
  try {
    add(23, 42)
  } with Logging {






  };
}
```

Effects and Handlers

Effect handlers

```
def main() : Int / { Logging } = {
  try {
    add(23, 42)
  } with Logging {
    def logMessage(s) = {
      println("Log message: " ++ s);
      resume(s);
    }

  };
}
```

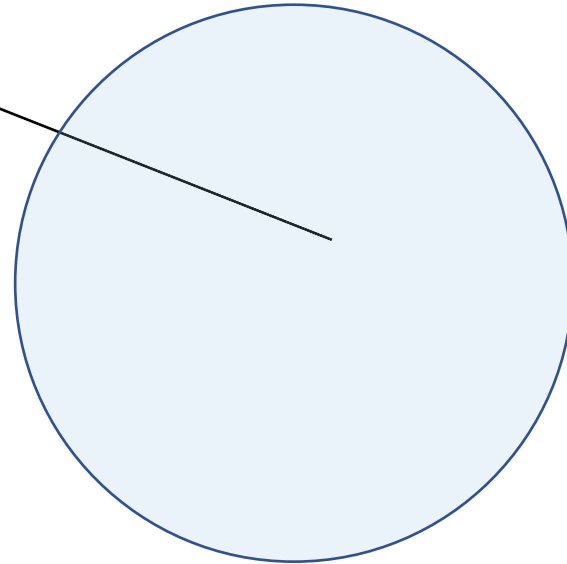
Effects and Handlers

Effect handlers

```
def main() : Int / {} = {
  try {
    add(23, 42)
  } with Logging {
    def logMessage(s) = {
      println("Log message: " ++ s);
      resume(s);
    }
    def logResult(s) = resume(s)
  };
}
```

IDE Support for Lexical Effects and Handlers

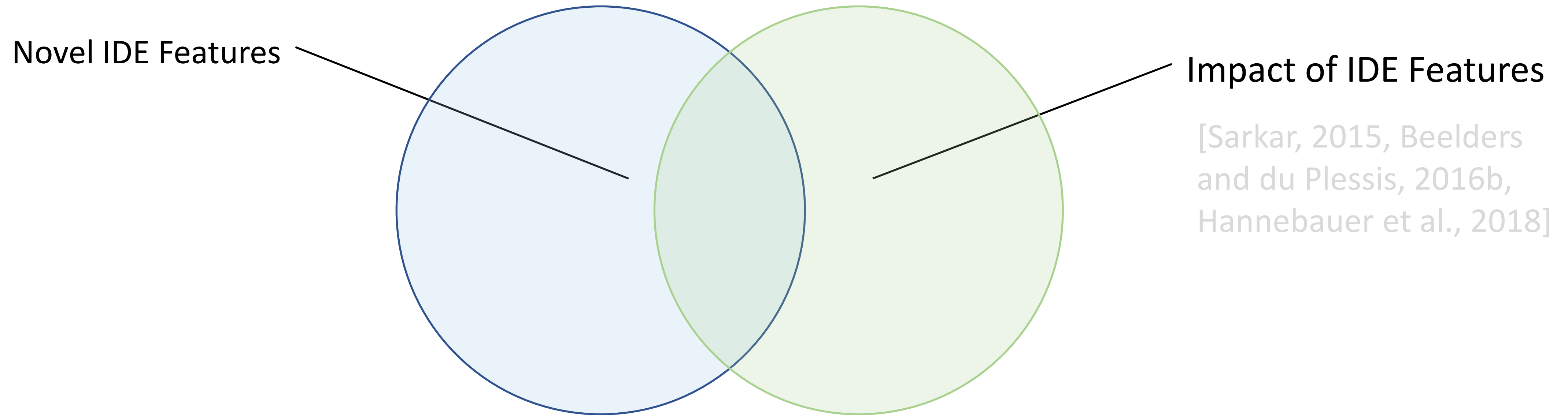
Research on Language Support in IDEs



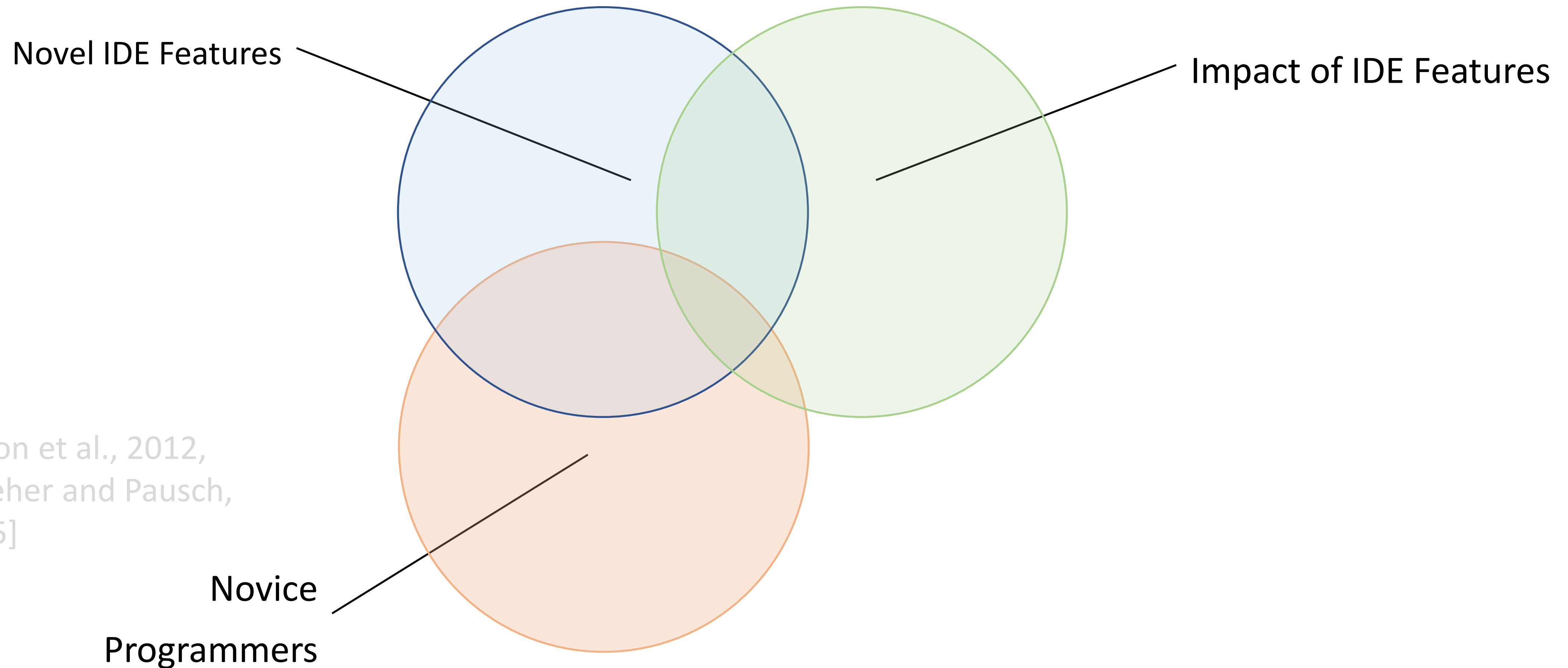
Novel IDE Features

[Fan et al., 2019,
Svyatkovskiy et al., 2019,
Rask et al.,2021]

Research on Language Support in IDEs

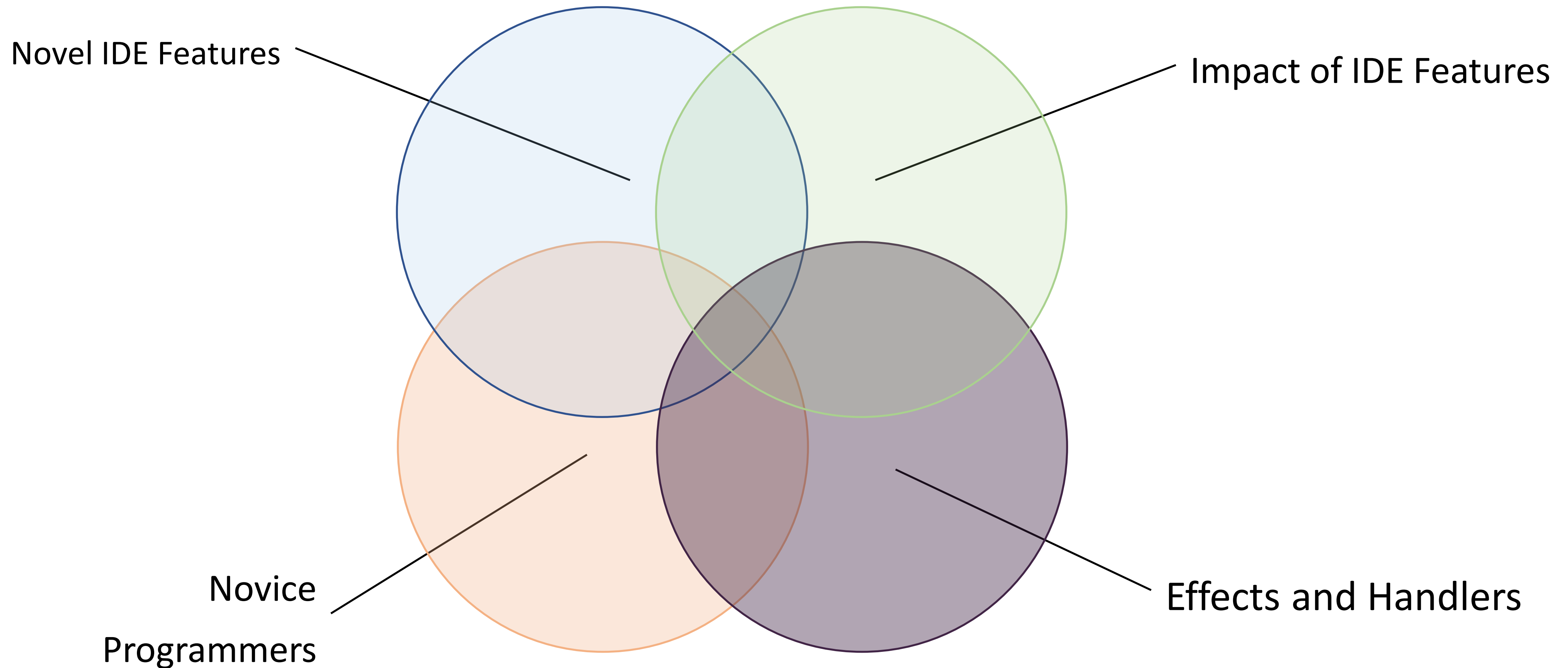


Research on Language Support in IDEs

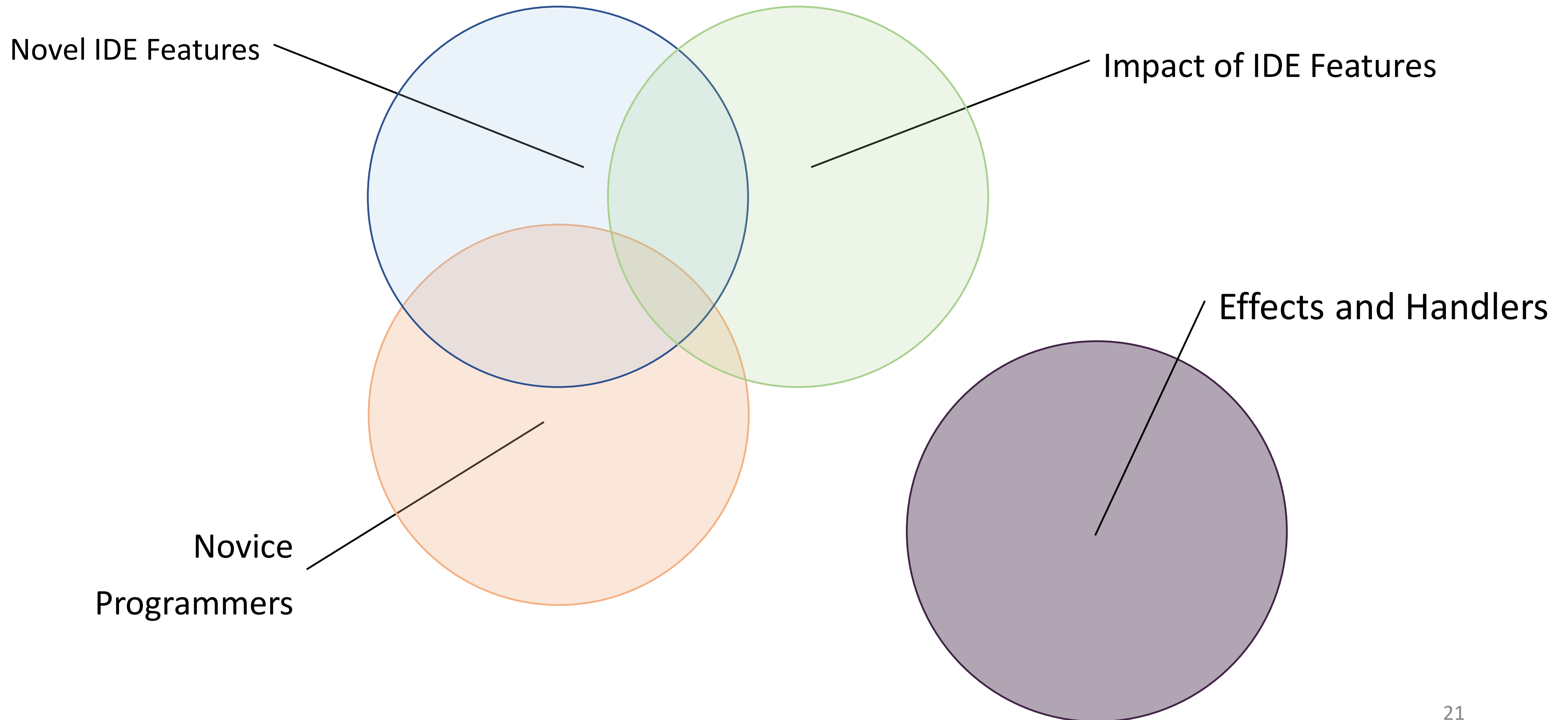


[Dillon et al., 2012, Kelleher and Pausch, 2005]

Research on Language Support in IDEs



Research on Language Support in IDEs



IDE Support for Effects and Handlers
-
Proposed features

Implemented Features

What we see

```
import SomeModule/Functions

def main() = {
  someFunction("foo")
}
```

Implemented Features

What we get

```
import SomeModule/Functions

def main() : String / { Logging } = {
  someFunction("foo")
}
```

Demo

ba_1.effekt

“the effect of static type systems is larger than often assumed, at least in comparison to code completion“.

[Fischer and Hanenberg, 2015]

"Using types helps improve readability" of source code.

[Meyerovich and Rabkin, 2013]

Implemented Features

Where does the Error effect come from?

```
def ex() : Int / { Fail, Next, Error, Flip } = {
    or {
        accept("do");
        commit {
            accept("foo");
            0;
        }
    } {
        accept("do");
        accept("bar");
        1
    }
}
```

Demo

ba_2.effekt

Similarity to jump-to-definition and find-usage.

Navigation in code: considered a frequently occurring aspect of programming, seen as an important skill of a programmer.

[Murphy et al., 2006, Jones and Burnett, 2007, Mader and Egyed, 2011]

Implemented Features

Handle the Error effect

```
def ex() : Int / { Fail, Next, Error, Flip } = {
    or {
        accept("do");
        commit {
            accept("foo");
            0;
        }
    } {
        accept("do");
        accept("bar");
        1
    }
}
```

Demo

ba_3.effekt

Similar to other code generating utilities:

auto-completion, refactoring tools, code snippet insertion.

[Murphy et al., 2006, Robbes and Lanza, 2008]

Lack of syntactical knowledge and coding errors: source of frustration among programmers.

[Rodrigo and Baker, 2009, Ford and Parnin, 2015]

Outlined Features

What is storeInCloud?

```
import Abstractions/Storage

def main() = {
  storeInCloud("foo")
}
```

Outlined Features

What is storeInCloud?

```
import Abstractions/Storage

def main() = {
  storeInCloud("foo")
}
```

An effect operation!

```
import Abstractions/Storage

def main() : Int / { Storage } = {
  (Storage => Storage.storeInCloud("foo"))
}
```

Outlined Features

What code requires the <u>Logging</u> effect?

```
def main() = {
    try {
        foo(42);
        bar(23){
            p => baz(p)
        }
    } with Logging {
        def logMessage(s) = resume(s)
    } with Error {
        def fail() = logMessage("Aborting!")
    }
}
```

Outlined Features

foo and baz require the Logging effect

```
def main() = {
    try {
        foo(42)(Logging, Error);
        bar(23)(Error){
            p => baz(p)(Logging)
        }
    } with Logging {
        def logMessage(s) = resume(s)
    } with Error {
        def fail() = logMessage("Aborting!")
    }
}
```

Secondary information in source code?

Code comments since 1960s.

[Sammet, 1978]

Natural language embedded in code:

source of misinterpretation.

[Van De Vanter, 2002]

Outlined Features

How would one utilise these effects?

```
effect Logging {
  def logString(s: String) : String
}

effect State {
  def get() : Int
  def put(n : Int) : Unit
}

effect Magic {
    def wizard() : rabbit
    def hat(r: rabbit) : Unit
}
```

Outlined Features

What is the Magic effect?

```
effect Magic {
    def wizard() : rabbit
    def hat(r: rabbit) : Unit
}
```

Outlined Features

What is the Magic effect?

```
effect Magic {
    def wizard() : rabbit
    def hat(r: rabbit) : Unit
}
```

An IDE could list available handler implementations

Outlined Features

It‘s just an analogy to the State effect

```
def magicShow { prog: Unit / Magic } = {
    var s = rabbit(0);
    try {
        prog()
    } with Magic {
        def wizard() = resume(s)
        def hat(r) = {
            s = r;
            resume(())
        }
    }
    s
}
```

Exemplary code:

considered helpful during programming

[Zagalsky et al.,2012, Nasehi et al., 2012]

Reading / understanding of foreign source code:

crucial aspect of software development

[Raymond, 1991, Busjahn and Schulte, 2013, Busjahn et al., 2014]

Outlined Features

You finished developing a library

```
effect logging {
  def logResult[A](s: A) : A
}

effect async {
  def fetch(s: String) : String
}

def add(a: Int, b: Int) = {
  val res = a + b;
  logResult(res)
}

def someLibraryFunction() = {
  try {
    add(23, 42)
  } with logging {
    def logResult(s) = {
      println("Logging result: " ++ s.show);
      resume(s);
    }
  };
}
```

Outlined Features

You finished developing a library

```
effect logging {
  def logResult[A](s: A) : A
}

effect async {
  def fetch(s: String) : String
}

def add(a: Int, b: Int) = {
  val res = a + b;
  logResult(res)
}

def someLibraryFunction() = {
  try {
    add(23, 42)
  } with logging {
    def logResult(s) = {
      println("Logging result: " ++ s.show);
      resume(s);
    }
  };
}
```

But you were lazy:

<u>No function is explicitly typed.</u>

Outlined Features

You finished developing a library

```
effect logging {
  def logResult[A](s: A) : A
}

effect async {
  def fetch(s: String) : String
}

def add(a: Int, b: Int) {
  val res = a + b;
  logResult(res)
}

def someLibraryFunction() {
  try {
    add(23, 42)
  } with logging {
    def logResult(s) = {
      println("Logging result: " ++ s.show);
      resume(s);
    }
  };
}
```

Make types & effects explicit
- Per function
- Per file
- Per project

You finished developing a library

```
effect logging {
  def logResult[A](s: A) : A
}

effect async {
  def fetch(s: String) : String
}

def add(a: Int, b: Int) : Int / logging = {
  val res = a + b;
  logResult(res)
}

def someLibraryFunction() : Int / Console = {
  try {
    add(23, 42)
  } with logging {
    def logResult(s) = {
      println("Logging result: " ++ s.show);
      resume(s);
    }
  };
}
```

Dynamic and static typing both offer benefits.

They may co-exist in programming languages.

[Meijer and Drayton, 2004].

Metals[1] language server for Scala:

“insert type annotation” adds explicit typing to implicitly typed expression.

[1]https://scalameta.org/metals

Conclusion

- Effects and handlers allow for novel IDE features
- These features could help in reasoning about and programming with effects and handlers
- Further research is needed to estimate the usefulness of presented features

Thank you!

References

[Beelders and du Plessis, 2016] Beelders, T. and du Plessis, J.-P. (2016). The influence of syntax highlighting on scanning and reading behaviour for source code. In Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists, pages 1–10.

[Brachthäuser et al., 2020] Brachthäuser, J. I., Schuster, P., and Ostermann, K.(2020). Effekt: Lightweight effect polymorphism for handlers. Technical report, Technical Report. University of Tübingen, Germany.

[Busjahn et al., 2014] Busjahn, T., Bednarik, R., and Schulte, C. (2014). What in-fluences dwell time during source code reading? analysis of element type and frequency as factors. In Proceedings of the Symposium on Eye Tracking Research and Applications, pages 335–338.

[Busjahn and Schulte, 2013] Busjahn, T. and Schulte, C. (2013). The use of code reading in teaching programming. In Proceedings of the 13th Koli Calling international conference on computing education research, pages 3–11.

[Dillon et al., 2012] Dillon, E., Anderson, M., and Brown, M. (2012). Comparing feature assistance between programming environments and their" effect" on novice programmers. Journal of Computing Sciences in Colleges, 27(5):69–77.

[Fan et al., 2019] Fan, H., Li, K., Li, X., Song, T., Zhang, W., Shi, Y., and Du, B.(2019). Covscode: A novel real-time collaborative programming environment for lightweight ide. Applied Sciences, 9(21).

[Fischer and Hanenberg, 2015] Fischer, L. and Hanenberg, S. (2015). An empirical investigation of the effects of type systems and code completion on api usability using typescript and javascript in ms visual studio. ACM SIGPLAN Notices,51(2):154–167.

[Ford and Parnin, 2015]Ford, D. and Parnin, C. (2015). Exploring causes of frustration for software developers. In2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering, pages 115–116. IEEE.

[Hannebauer et al., 2018] Hannebauer, C., Hesenius, M., and Gruhn, V. (2018). Does syntax highlighting help programming novices? Empirical Software Engineering,23(5):2795–2828.

[Jones and Burnett, 2007]Jones, S. J. and Burnett, G. E. (2007). Spatial skills and navigation of source code.ACM SIGCSE Bulletin, 39(3):231–235.

[Kelleher and Pausch, 2005] Kelleher, C. and Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv., 37(2):83–137.

[Mader and Egyed, 2011]Mader, P. and Egyed, A. (2011). Do software engineers benefit from source code navigation with traceability? – an experiment in software change management. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE ’11, page 444–447, USA. IEEE Computer Society.

[Meijer and Drayton, 2004] Meijer, E. and Drayton, P. (2004). Static typing wherepossible, dynamic typing when needed: The end of the cold war between programming languages. Citeseer.

References

[Meyerovich and Rabkin, 2013] Meyerovich, L. A. and Rabkin, A. S. (2013). Empiri-cal analysis of programming language adoption. SIGPLAN Not., 48(10):1–18.

[Murphy et al., 2006]Murphy, G. C., Kersten, M., and Findlater, L. (2006). How are java software developers using the elipse ide? IEEE software, 23(4):76–83.

[Nasehi et al., 2012] Nasehi, S. M., Sillito, J., Maurer, F., and Burns, C. (2012). What makes a good code example?: A study of programming q&a in stackoverflow.In2012 28th IEEE International Conference on Software Maintenance (ICSM), pages25–34. IEEE.

[Plotkin and Power, 2003] Plotkin, G. and Power, J. (2003).Algebraic operations and generic effects. Applied categorical structures, 11(1):69–94.

[Plotkin and Pretnar, 2009] Plotkin, G. and Pretnar, M. (2009).Handlers of algebraic effects. In European Symposium on Programming, pages 80–94. Springer

[Rask et al., 2021]Rask, J. K., Madsen, F. P., Battle, N., Macedo, H. D., and Larsen,P. G. (2021). Visual studio code vdm support. John Fitzgerald, Tomohiro Oda, and Hugo Daniel Macedo (Editors), page 35.

[Raymond, 1991]Raymond, D. R. (1991). Reading source code. In Proceedings of the1991 conference of the Centre for Advanced Studies on Collaborative research, pages3–16.

[Robbes and Lanza, 2008] Robbes, R. and Lanza, M. (2008). How program history can improve code completion. In2008 23rd IEEE/ACM International Conference on Automated Software Engineering, pages 317–326. IEEE.

[Rodrigo and Baker, 2009]Rodrigo, M. M. T. and Baker, R. S. (2009). Coarse-grained detection of student frustration in an introductory programming course. In Proceedings of the fifth international workshop on Computing education research workshop, pages 75–80.

[Sammet, 1978] Sammet, J. E. (1978). The early history of cobol. In History of Programming Languages, pages 199–243.

[Sarkar, 2015] Sarkar, A. (2015). The impact of syntax colouring on program com-prehension. In PPIG, page 8.

[Svyatkovskiy et al., 2019] Svyatkovskiy, A., Zhao, Y., Fu, S., and Sundaresan, N.(2019). Pythia: Ai-assisted code completion system. In Proceedings of the 25th ACMSIGKDD International Conference on Knowledge Discovery amp; Data Mining, KDD'19, page 2727–2735, New York, NY, USA. Association for Computing Machinery.

[Van De Vanter, 2002] Van De Vanter, M. L. (2002). The documentary structure of source code. Information and Software Technology, 44(13):767–782. Special Issue onSource Code Analysis and Manipulation (SCAM)

[Zagalsky et al., 2012] Zagalsky, A., Barzilay, O., and Yehudai, A. (2012). Example overflow: Using social media for code recommendation. In2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE), pages 38–42.IEEE.