The Semantic Workspace: Augmenting Exploratory Programming with Integrated Generative AI Tools

Defense of Master's Thesis

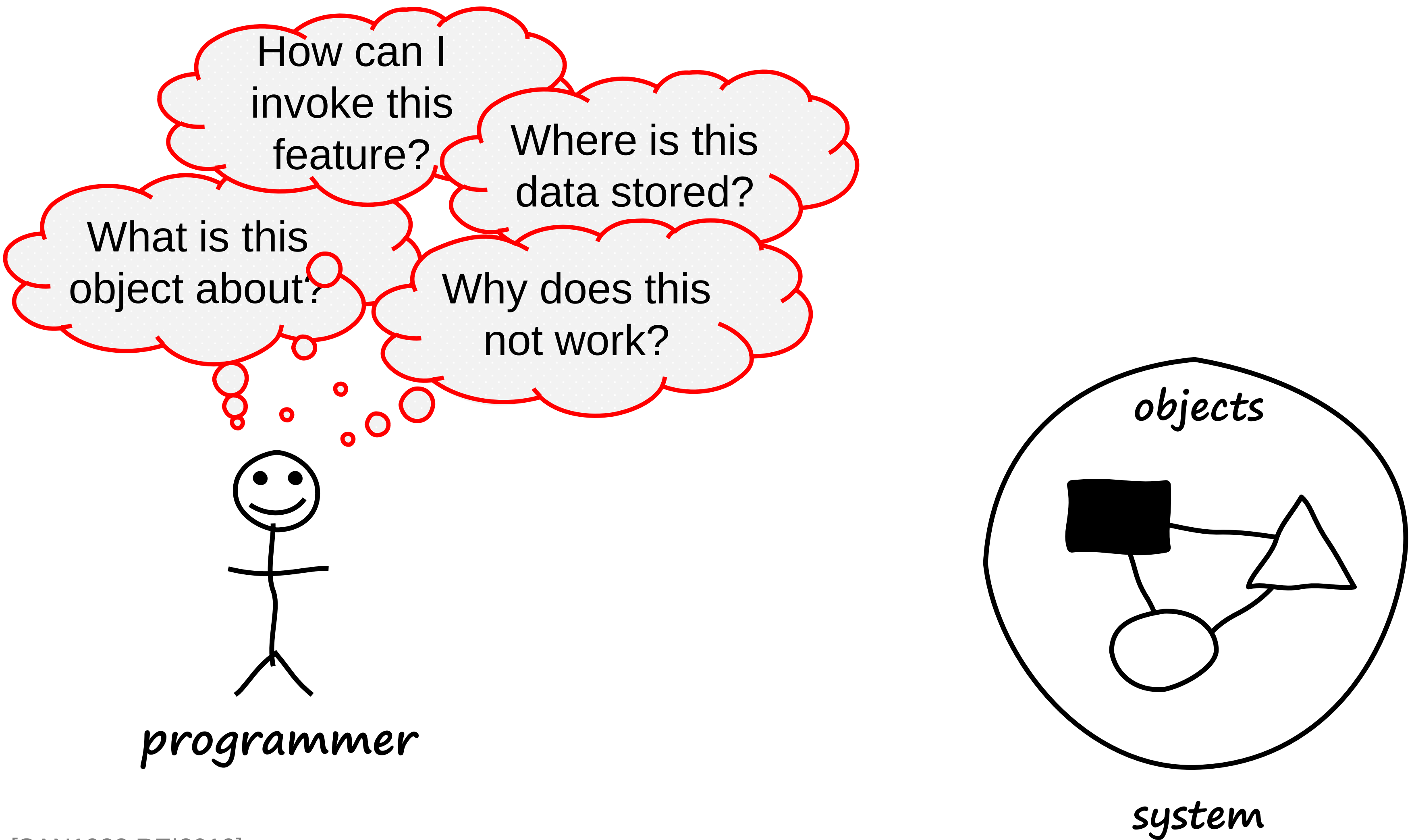
Christoph Thiede

HPI Software Architecture Group

Advisors: Robert Hirschfeld,
Marcel Taeumel, Lukas Böhme

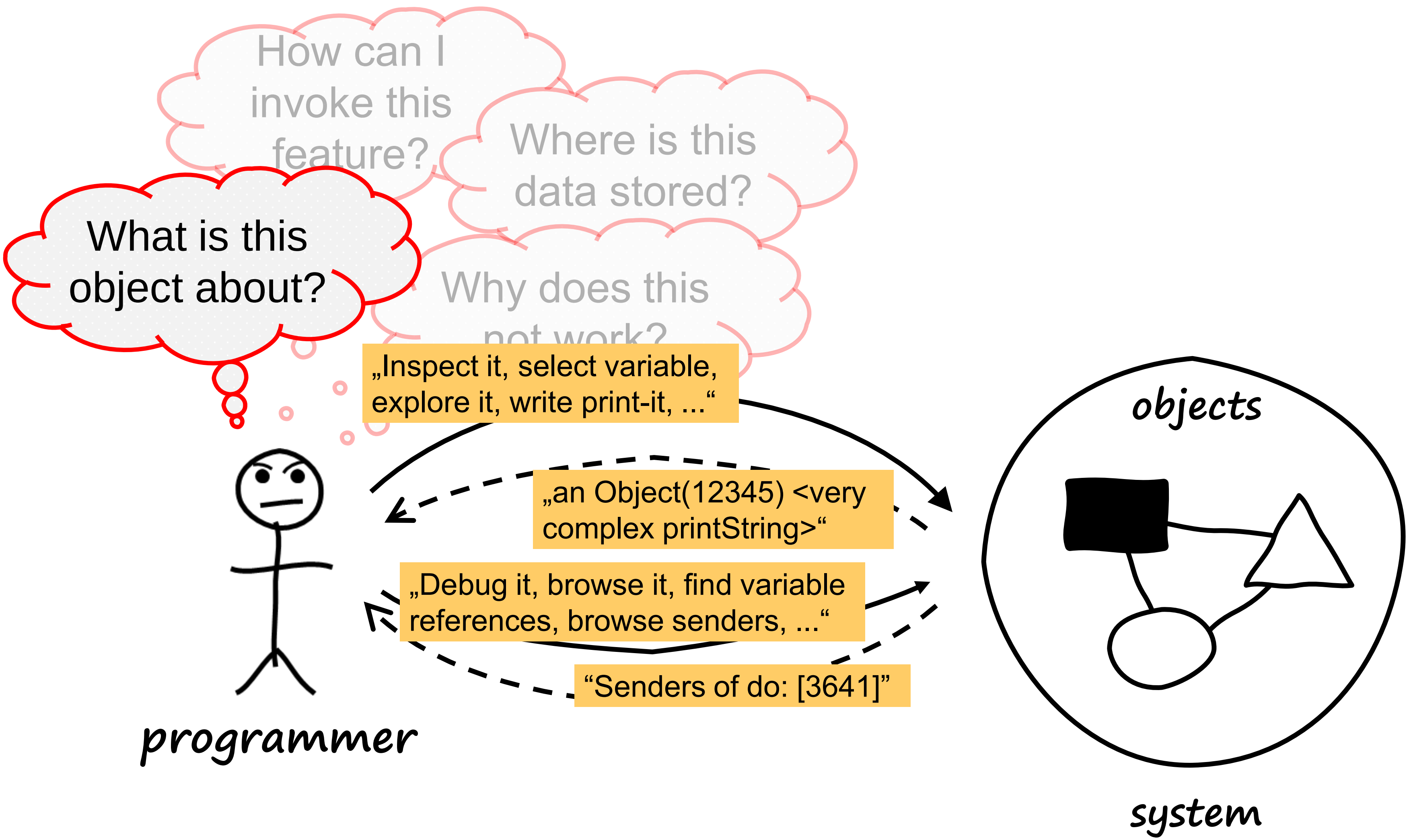
2024-12-06

Motivation

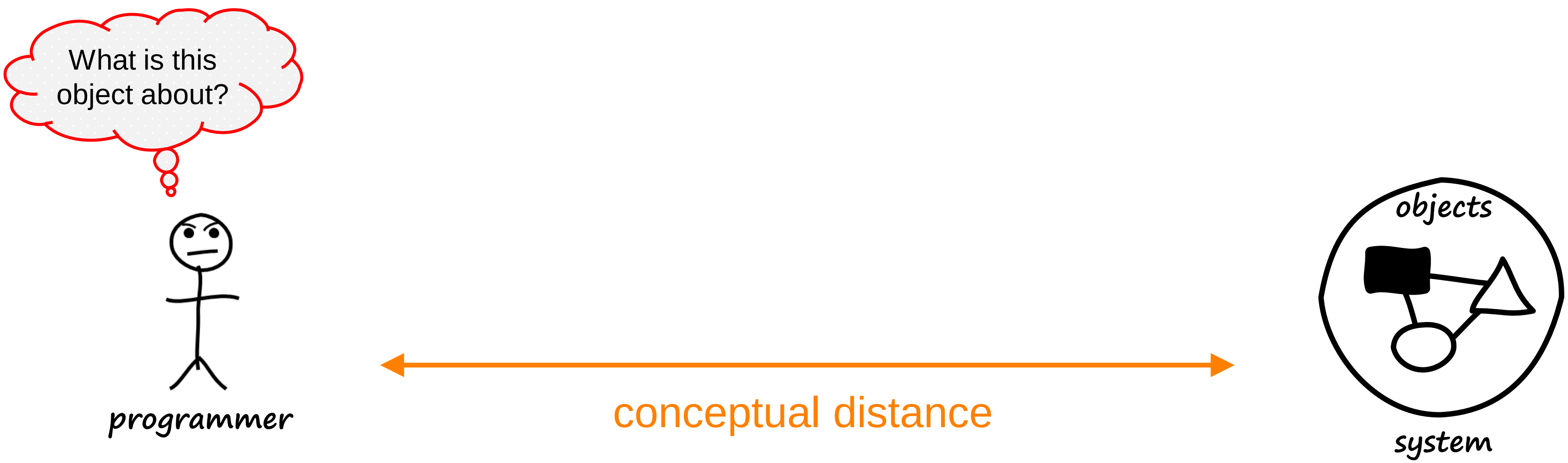


[SAN1988,REI2019]

Motivation

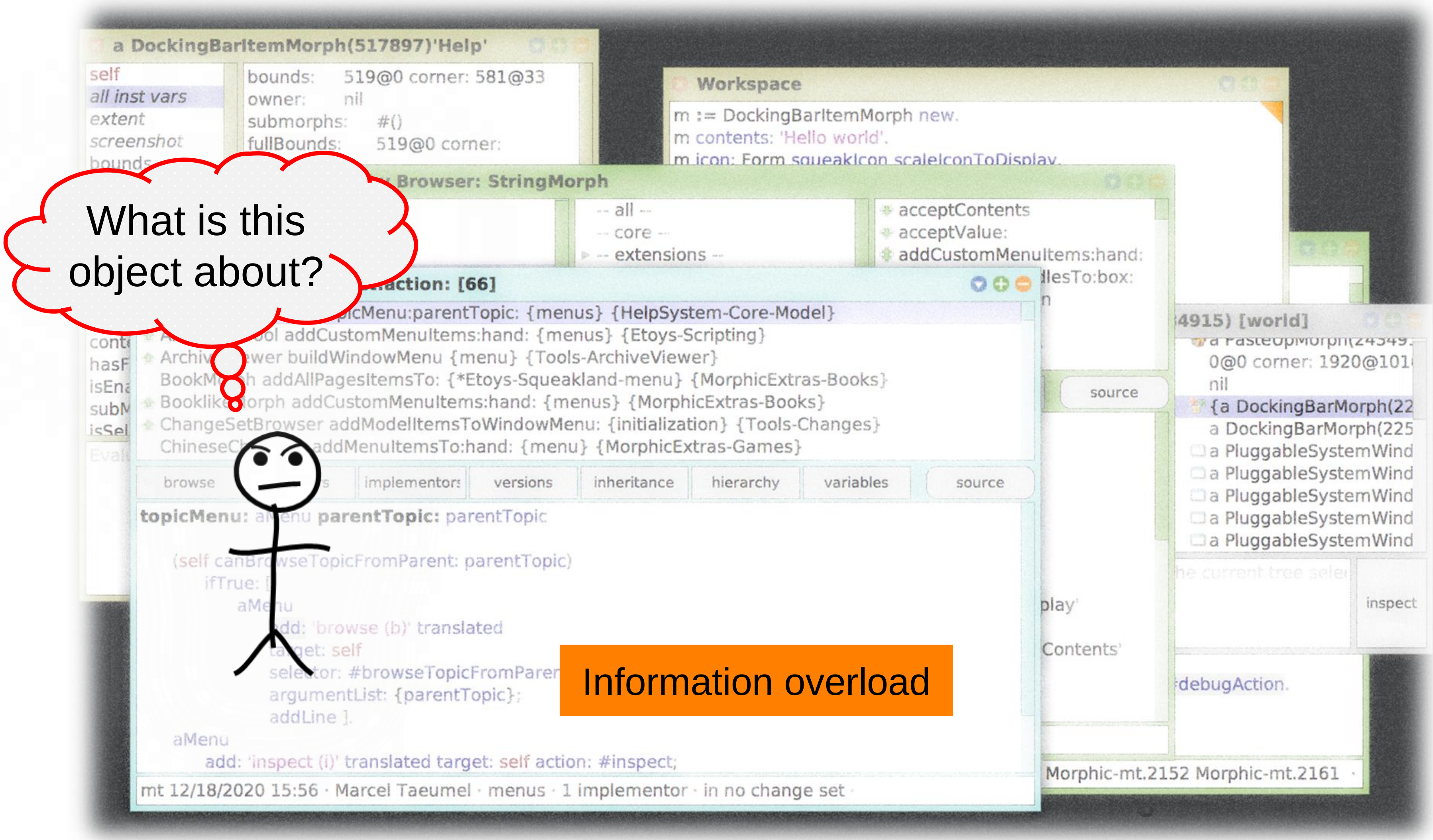

How can I invoke this feature?
Where is this data stored?
What is this object about?
Why does this not work?
„Inspect it, select variable, explore it, write print-it, ...“
„an Object(12345) <very complex printString>“
„Debug it, browse it, find variable references, browse senders, ...“
“Senders of do: [3641]”
objects
programmer
system

Motivation



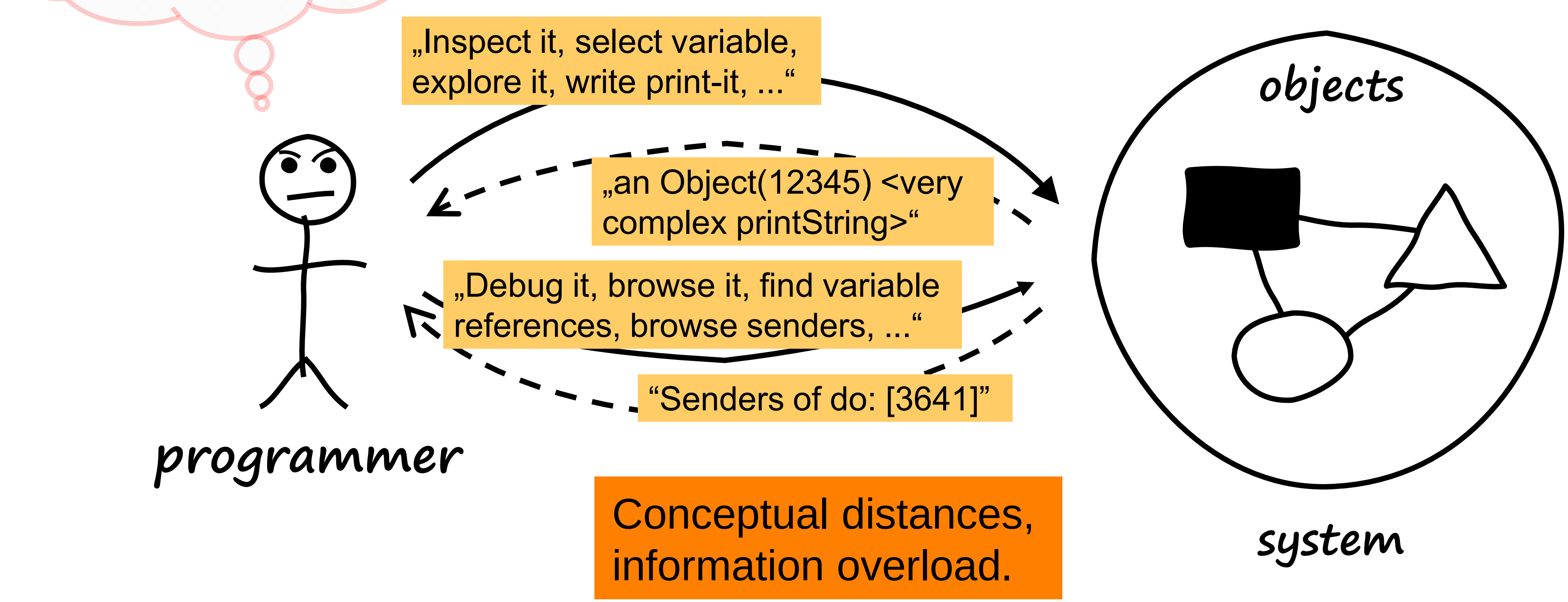
[UNG1997]

Motivation



Idea

- Semantic technologies already support programmers at different coding and comprehension tasks ...
  - LLMs generate and explain code
  - Embeddings allow for source code recommendations
- Why not use them to streamline and augment exploratory programming?

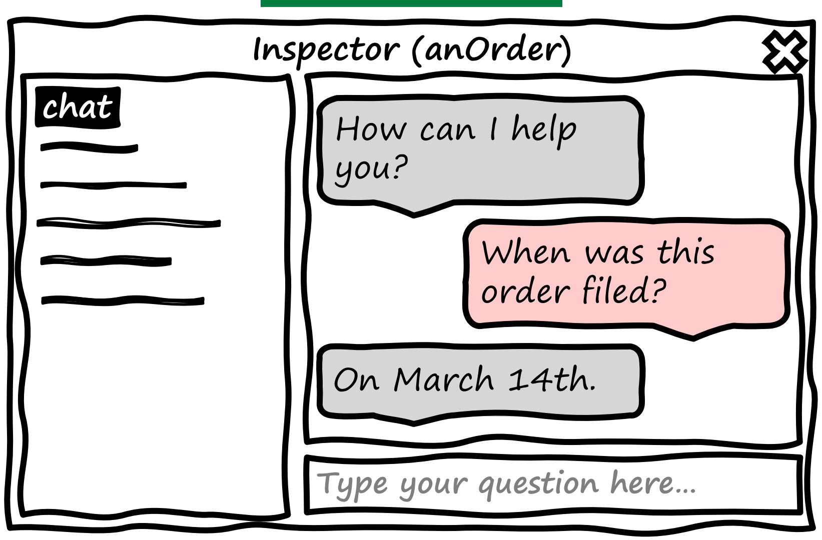


Idea

- Semantic technologies already support programmers at different coding and comprehension tasks ...
  - LLMs generate and explain code
  - Embeddings allow for source code recommendations
- Why not use them to streamline and augment exploratory programming?

STATE OF THE ART

OUR VISION



Research Question

How can we augment
the exploratory programming workflow
with semantic technologies?

LLMs +
embeddings

Thesis Statement

How can we augment
the exploratory programming workflow
with semantic technologies?

- We integrate semantic interfaces into exploratory programming systems
- Exploratory programmers delegate work to intelligent agents and collaborate with semantic technologies
- This augments and streamlines their exploratory programming workflow

Outline

- Intro
- Background
  - Exploratory Programming
  - Semantic Technologies
- Solution
  - Approach: Semantic Exploratory Programming
    - Semantic Suggestions
    - Semantic Completions
    - Semantic Conversations
  - Design: Semantic Exploration Kernel
    - Suggestion Engine
    - Semantic Object Interfaces
- Demo
- Discussion
  - Semantic Technologies
  - Exploratory Programming Experiences
- Related Work
- Conclusion & Future Work

Background

Background: Exploratory Programming

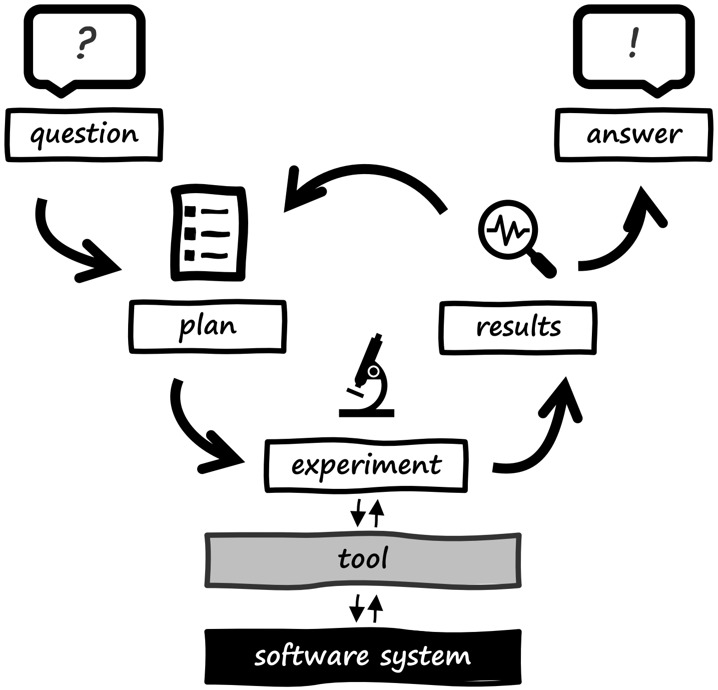
- Exploratory programmers understand and solve problems simultatenously and iteratively [SAN1988,REI2019]
- They conduct vivid and extensive conversations with systems through many small experiments [TAE2022]

Background: Exploratory Programming

- Our model: Exploratory programming is an iterative research process
  - Programmers formulate questions, conduct experiments, and evaluate results

    „When was this order created?“
  - Tools provide access to the software system

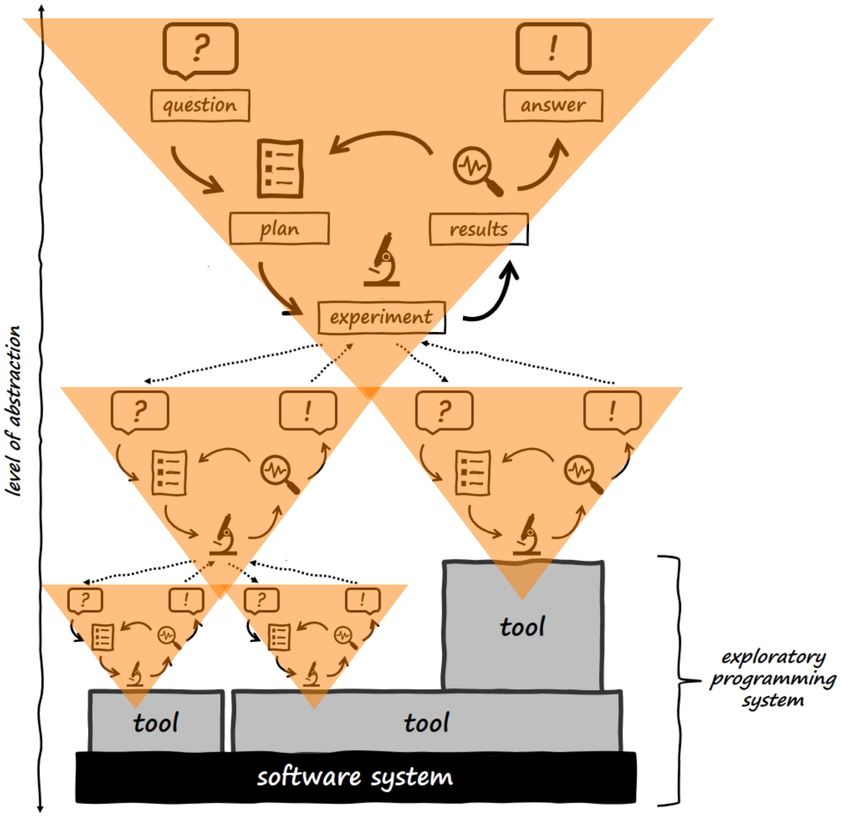
    Display variables of an object, browse methods of a class, debug processes...



[GOL1984,SAN1988]

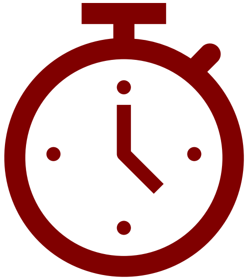
Background: Exploratory Programming

- Our model: Exploratory programming is an iterative nested research process
  - High-level experiments raise lower-level questions
    „How can I build this prototype?“
  - Higher-level tools facilitate access to software artifacts
    Task- and domain-specific interfaces (e.g., system browsers, visualizations)



[GOL1984,SAN1988,CHI2015]

Background: The Experience of Immediacy



Temporal immediacy

“Human beings **recognize causality** without conscious effort only when the **time between causally related events** is kept to a minimum.”

Spatial immediacy

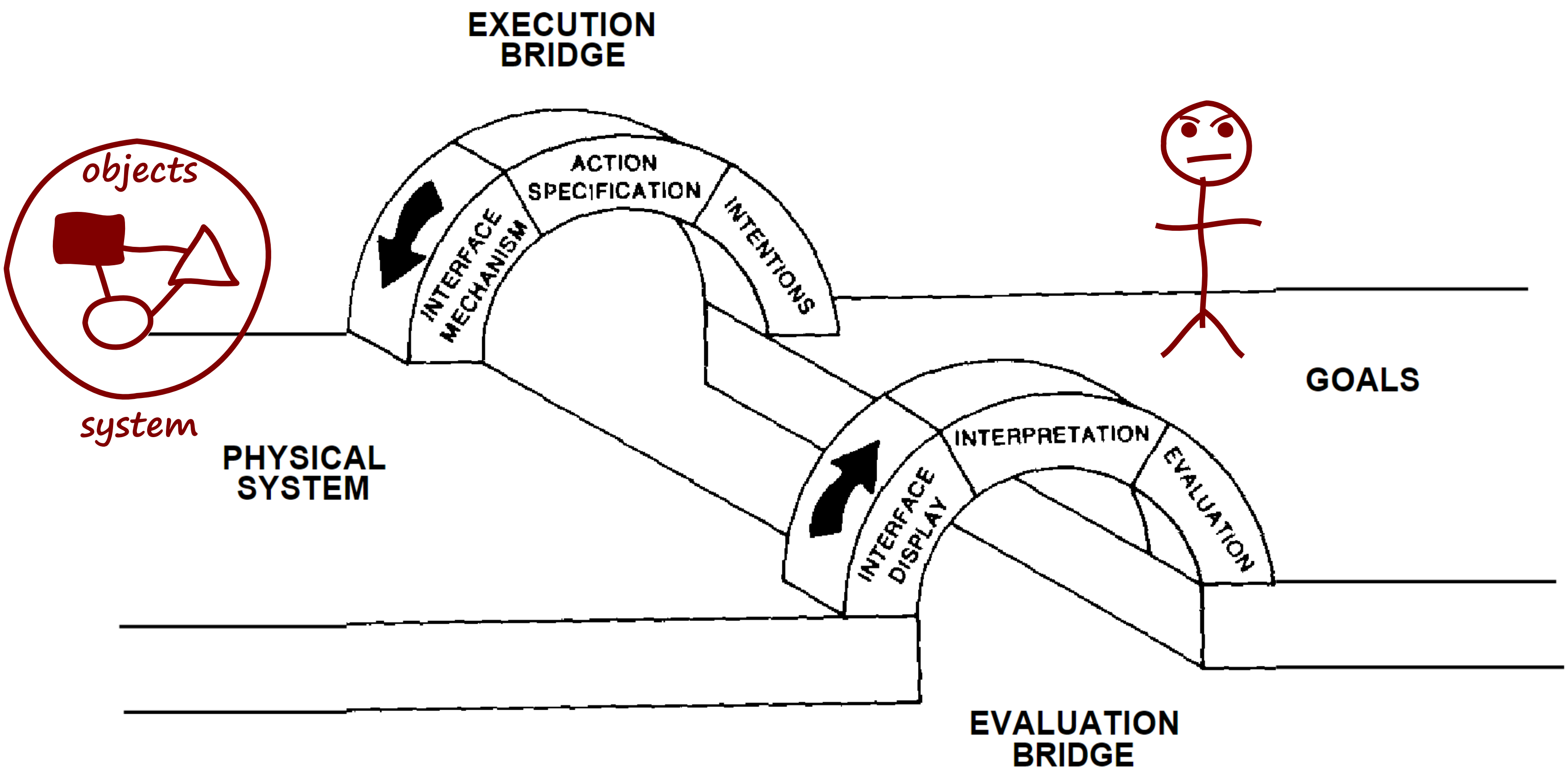
“[...] the **physical distance** between causally related events is kept to a minimum.”

Semantic immediacy

“[...] the **conceptual distance** between semantically related pieces of information is kept to a minimum.”

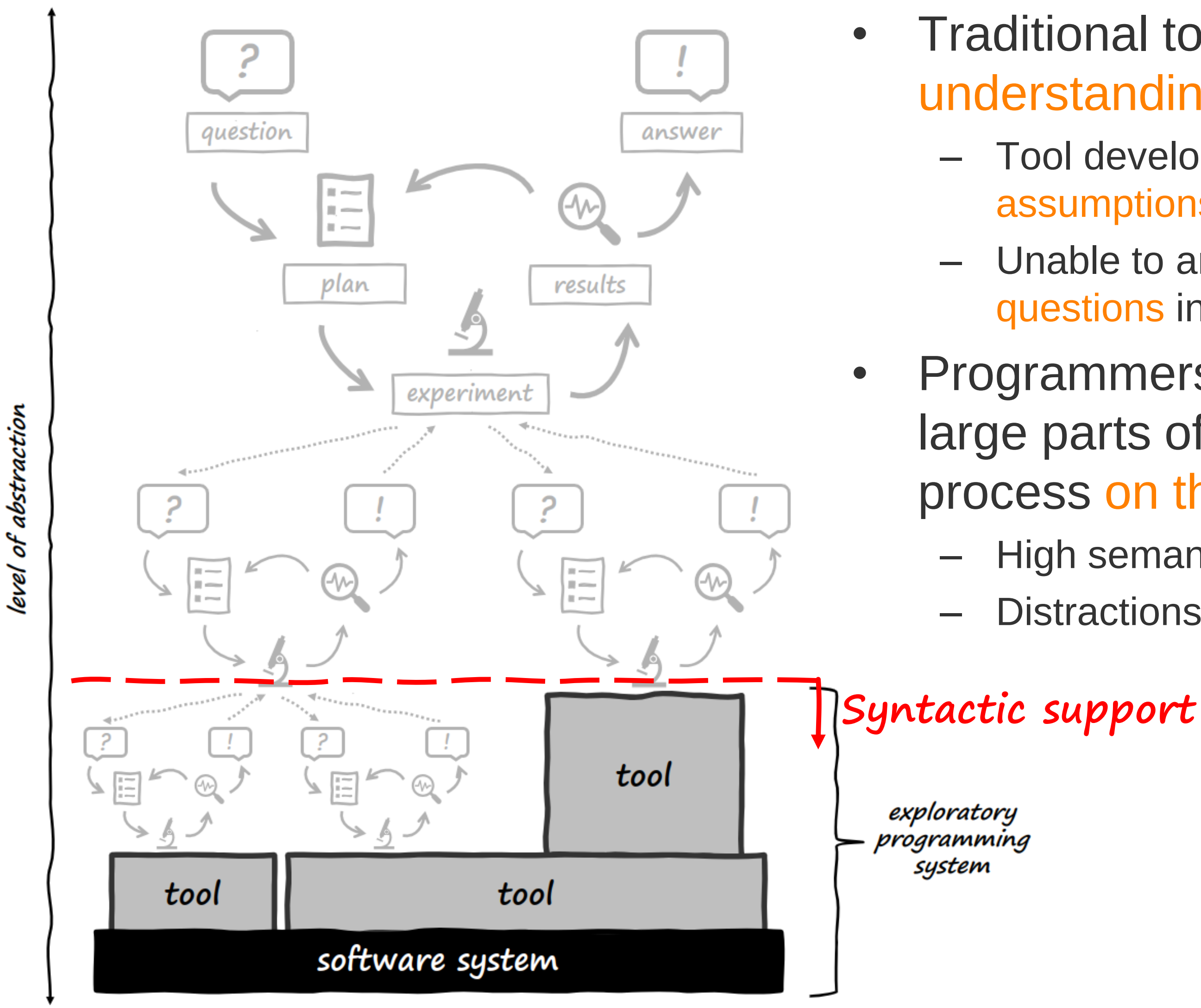
[UNG1997]

Background: Two Gulfs of HCI



[NOR1986]

Challenges in Exploratory Programming Systems: Limited Level of Abstraction



- Traditional tools have no understanding of tasks
  - Tool developers hard-code rigid assumptions
  - Unable to answer higher-level questions in unique context
- Programmers have to handle large parts of the research process on their own
  - High semantic distances
  - Distractions and cognitive load

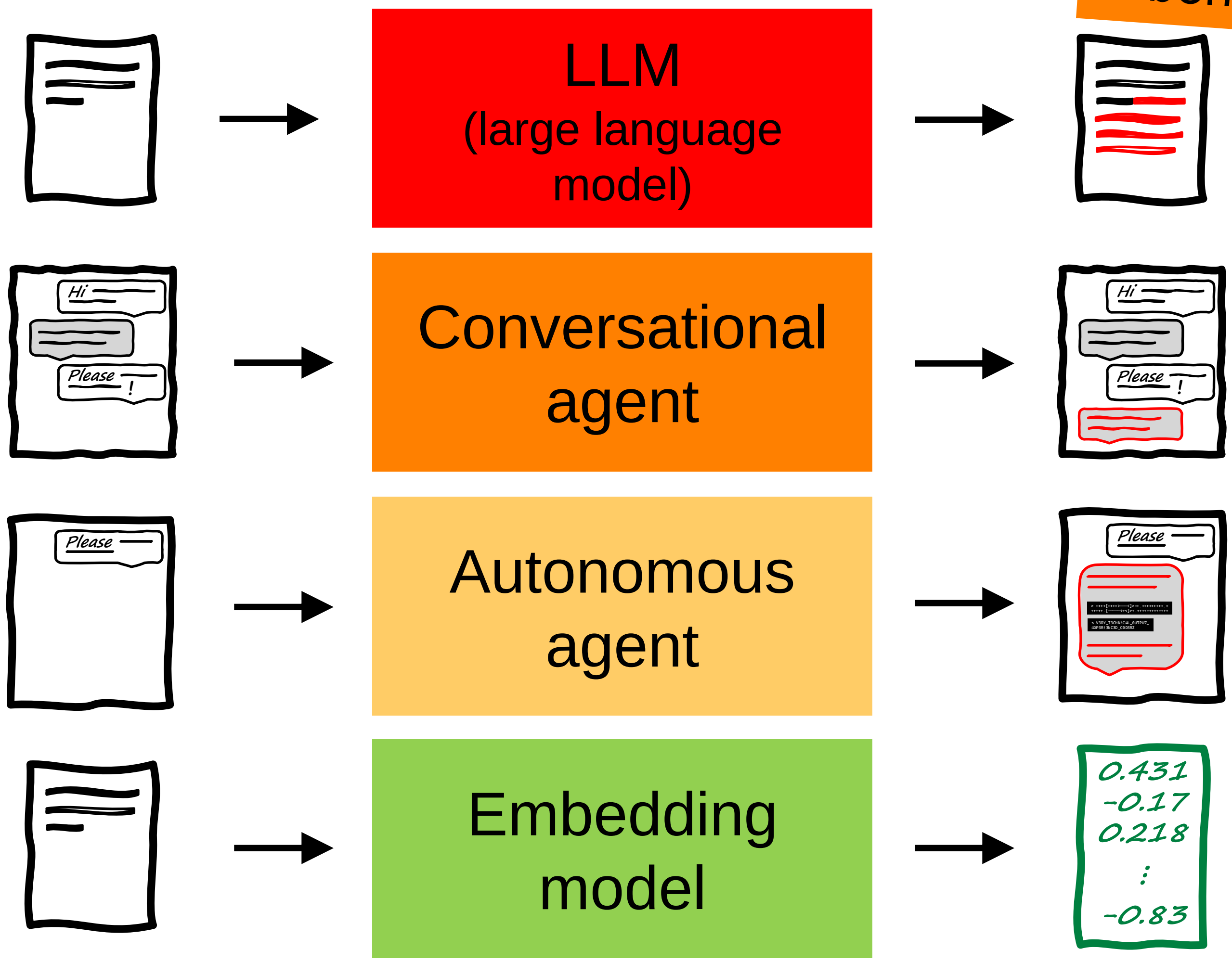
[CSI2008]

Background: Semantic Technologies

- Machine learning models that focus on the meaning (semantics) of text
  - Large language models (LLMs) generate and complete text [RAD2018,CHEN2021,WAY2023]
    - Source code, natural language, …
  - Conversational agents chat with user, call system functions, and reason autonomously [LEW2020,WAY2023]
  - Embedding models map objects to vectors to compare, search, and cluster them based on common concepts [MIK2013,DEV2019]

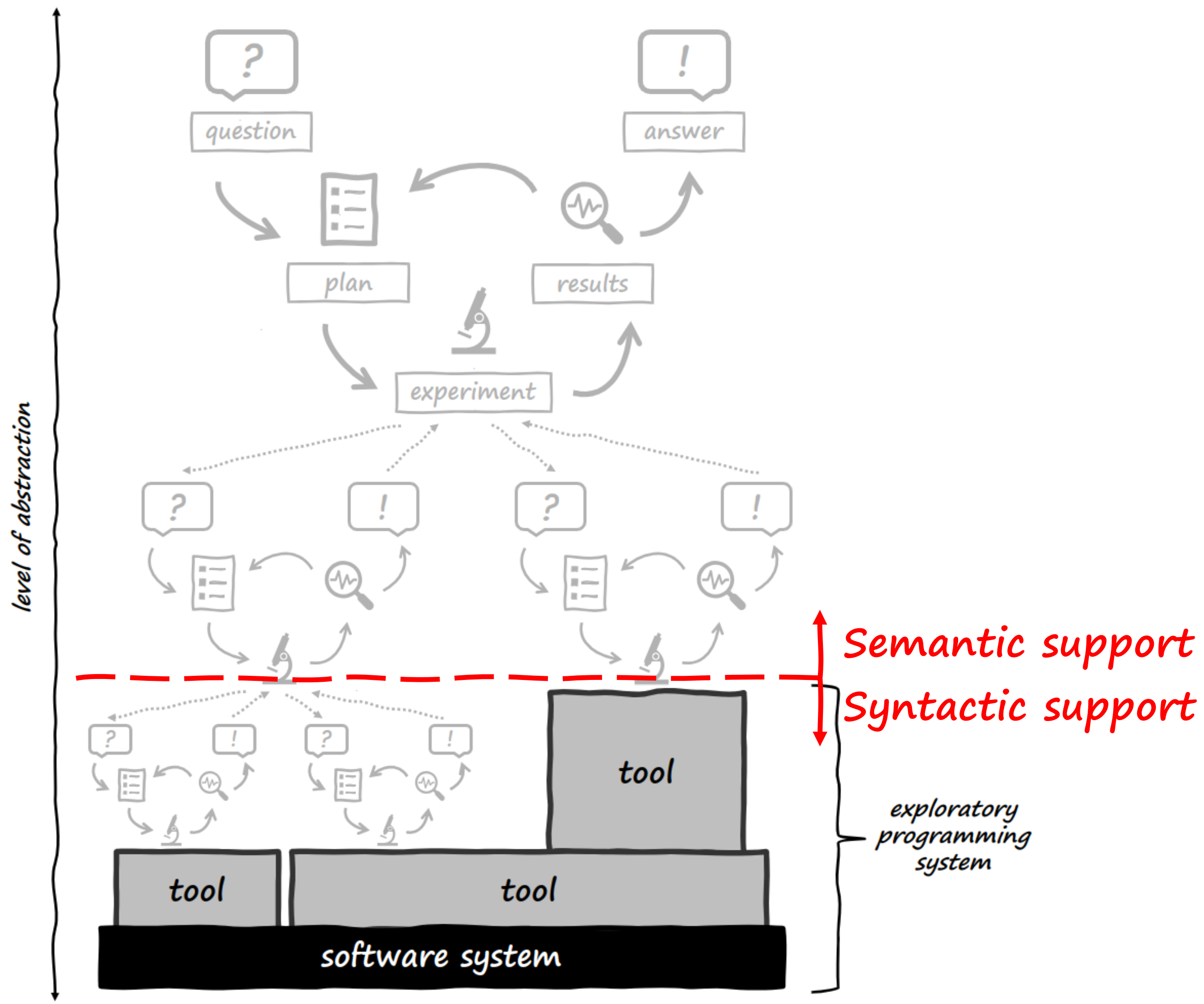
Background: Semantic Technologies

bonus content



[RAD2018,CHEN2021,BAR2023,WAY2023,LEW2020,MIK2013,DEV2019]

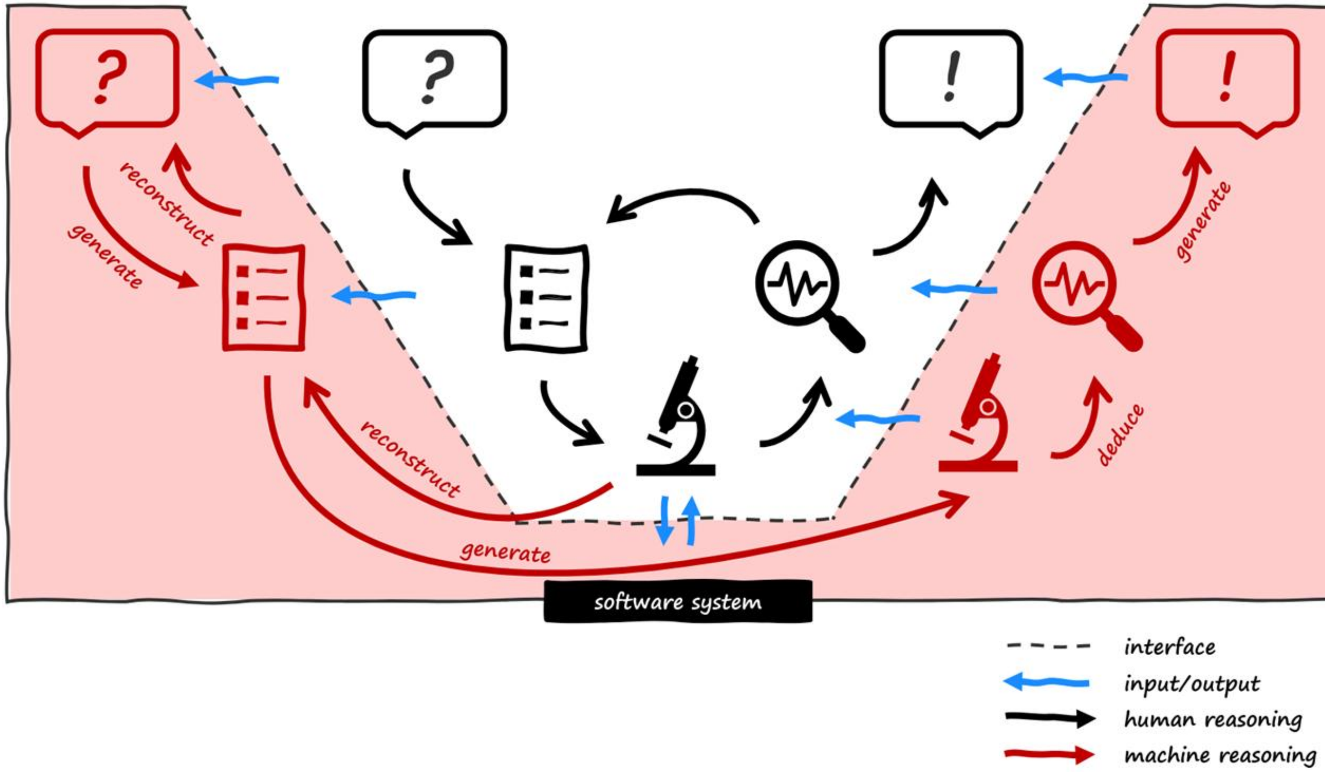
Opportunity: More Conceptual Support through Semantic Technologies



Approach

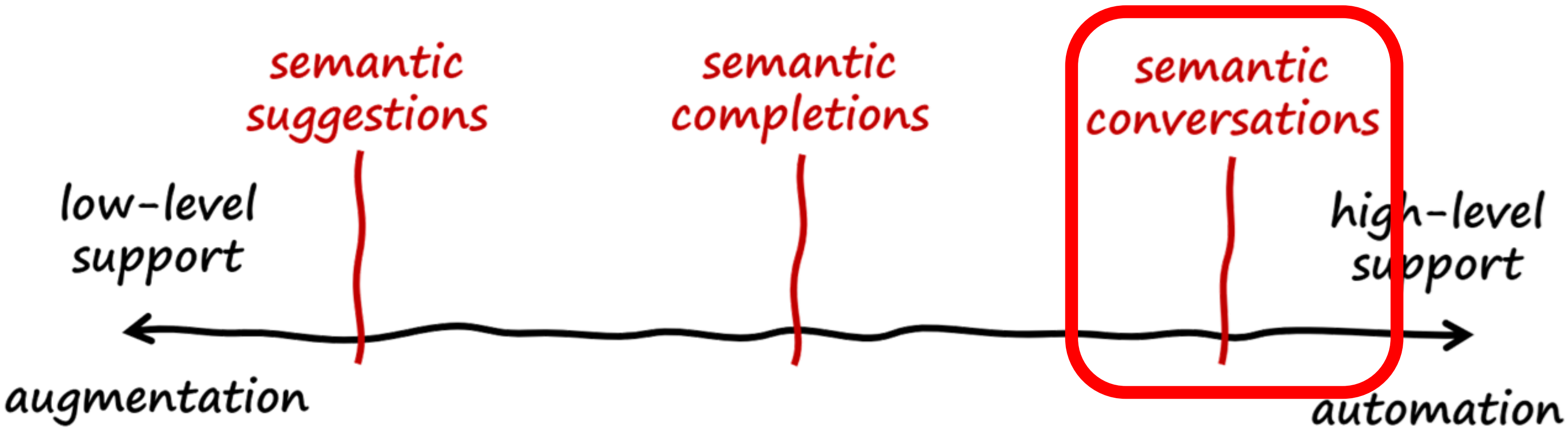
Approach: Augmented Exploratory Programming

- Programmers exchange conceptual artifacts with a semantic exploratory programming system through high-level interfaces
- Semantic system continues research process and suggests further artifacts



Approach: The Semantic Workspace

- Our conceptual framework of an exploratory programming system with different semantic tools
- Support spectrum:
  - Lower-level tools augment the research process with suggestions
  - Higher-level tools automate the research process

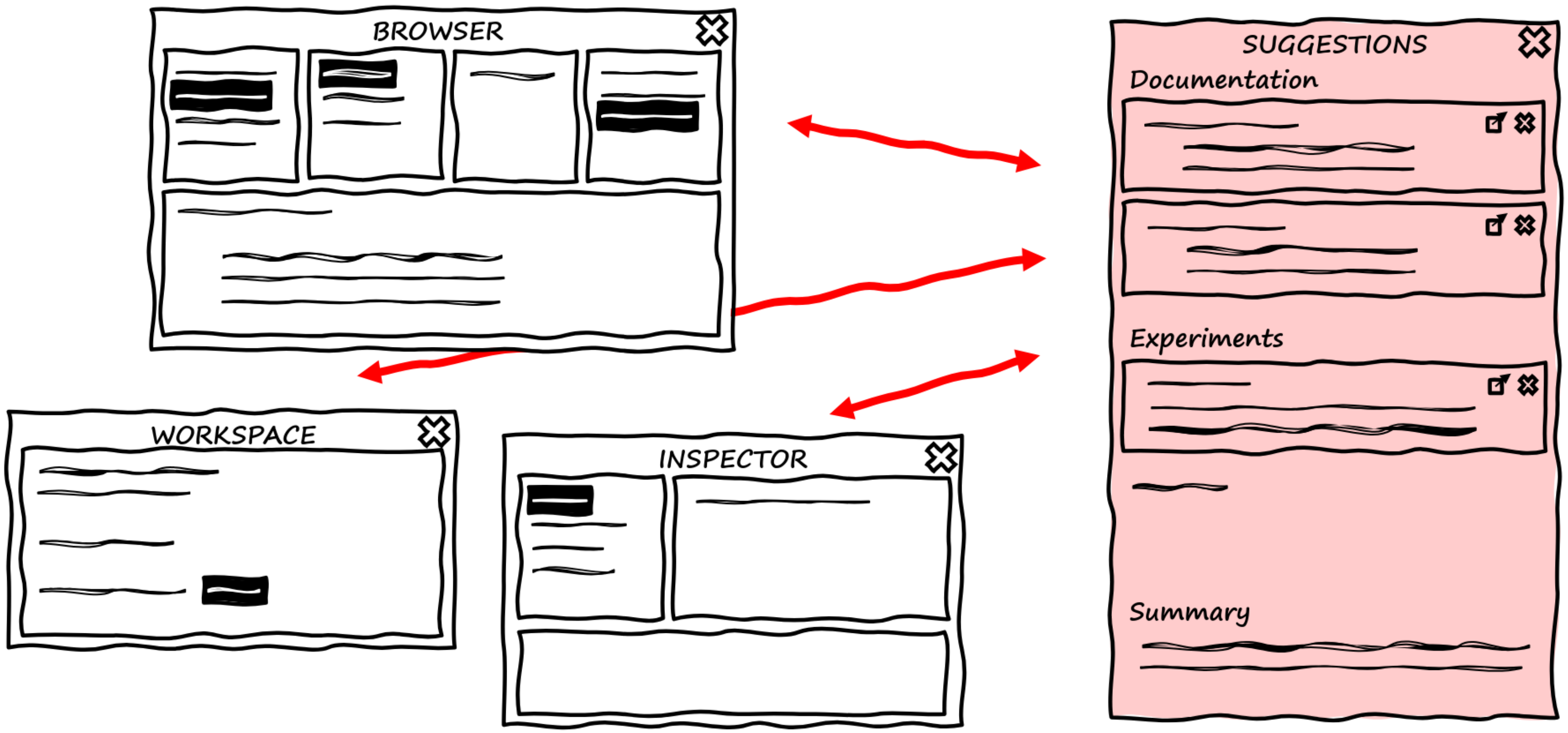


Approach: The Semantic Workspace

Semantic Suggestions

bonus content

- Track the experiments of programmers in traditional tools
- Anticipate their plans and next steps
- Suggest and summarize further experiments

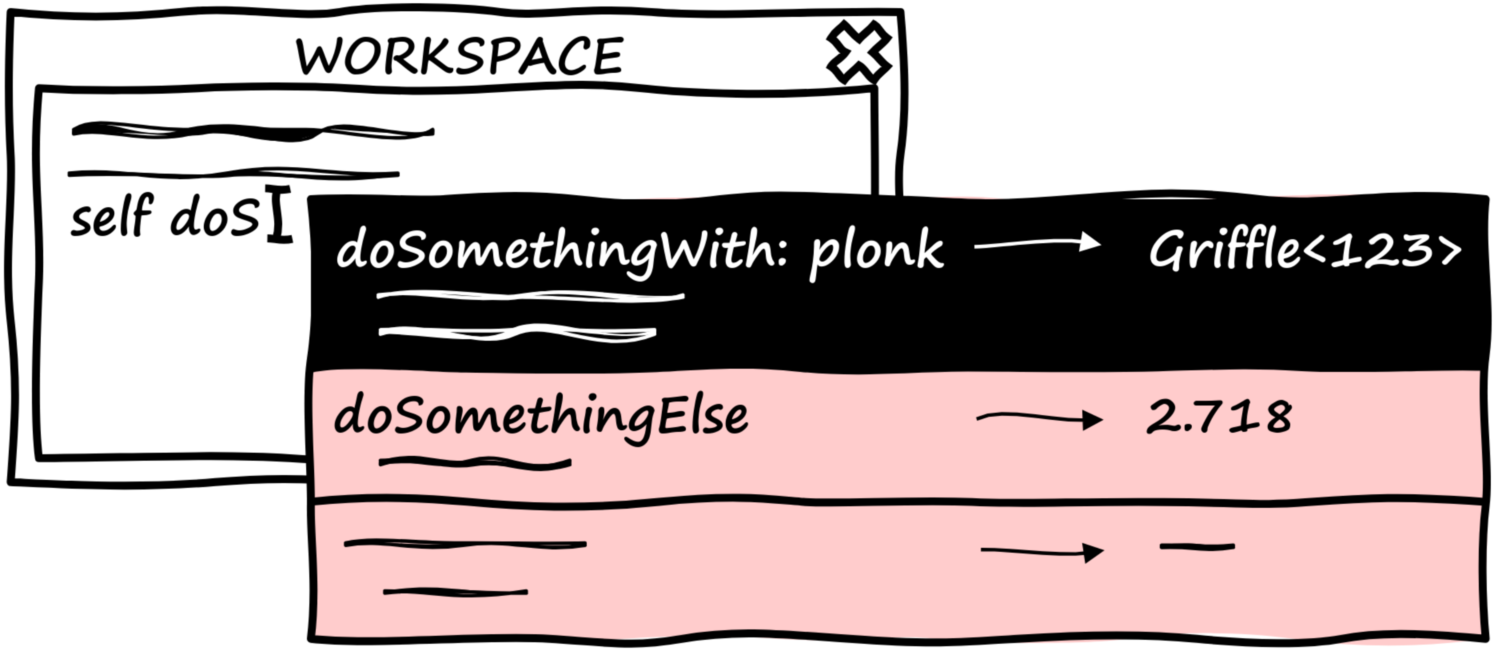


Approach: The Semantic Workspace

Semantic Completions

bonus content

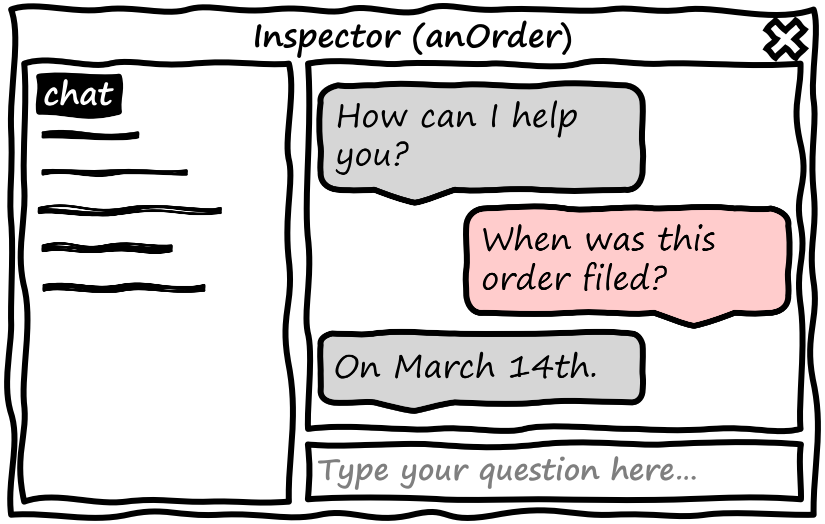
- Track planning activities of programmers (e.g., typing a script or method)
- Anticipate intentions and run possible experiments
- Suggest contextualized experiments by completing plans



Approach: The Semantic Workspace

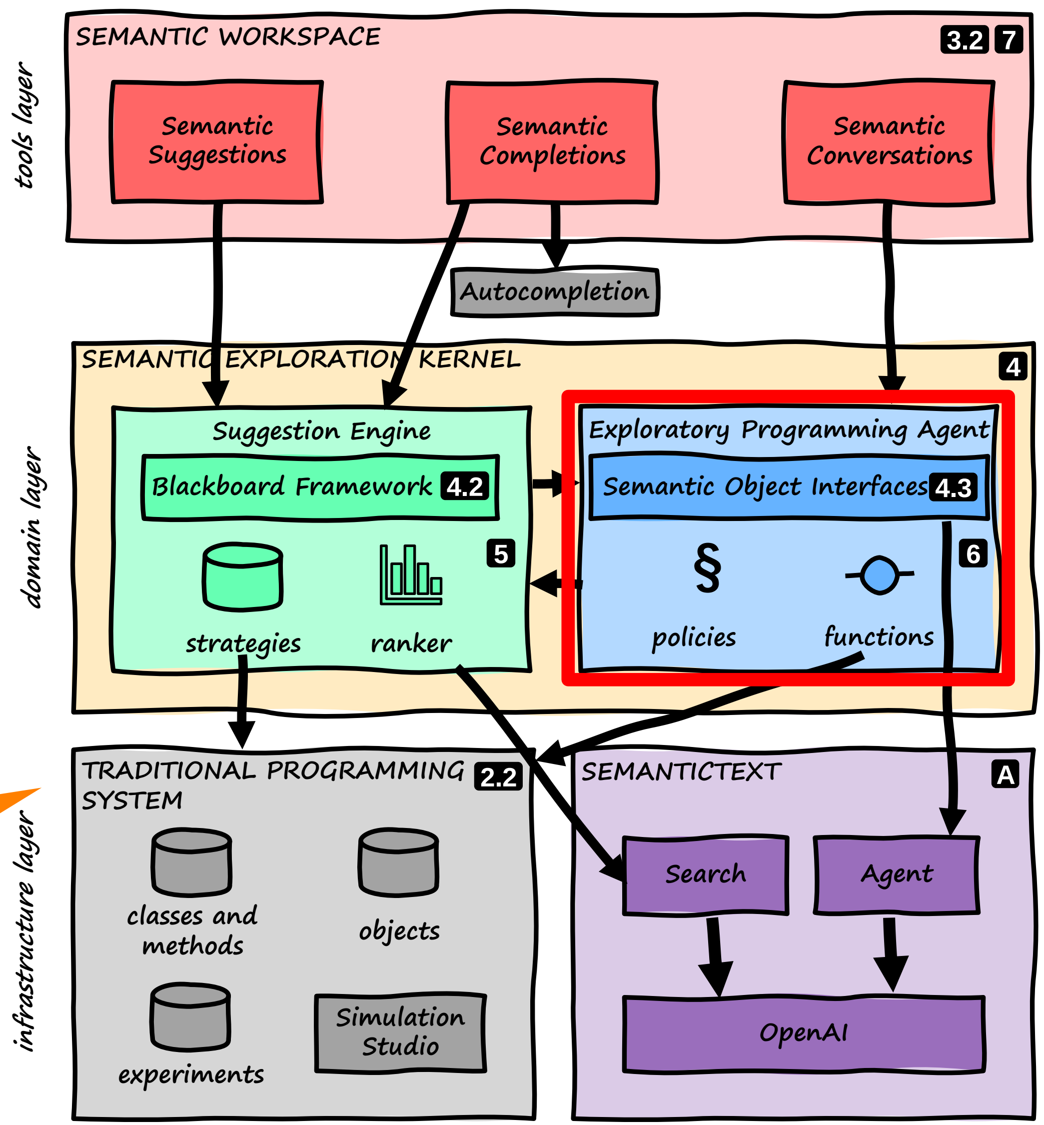
Semantic Conversations

- Answer conceptual questions about objects in natural language
- Autonomously conduct required research process



Design

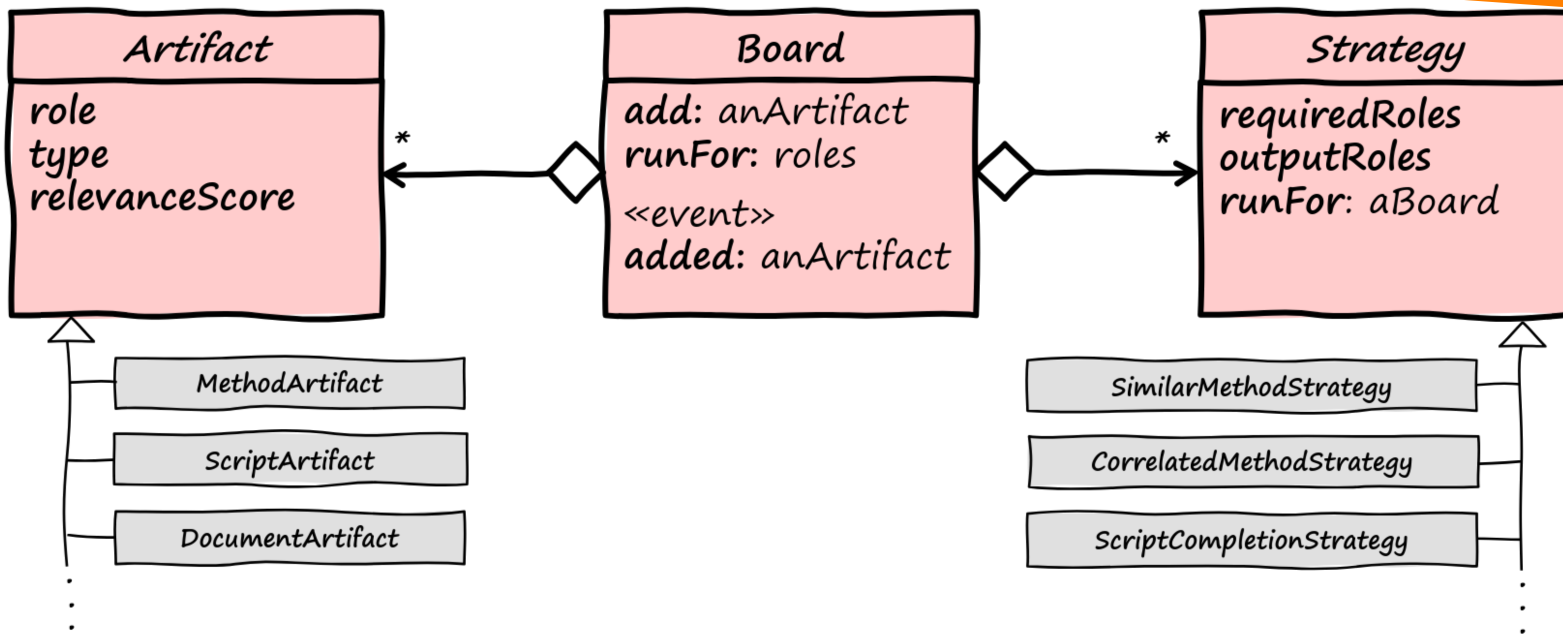
Design

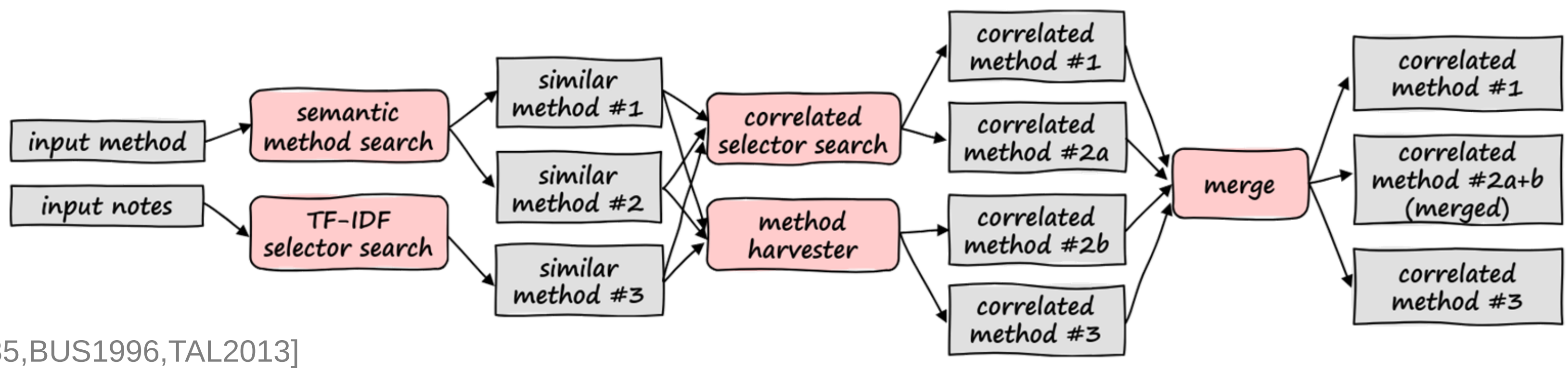


Existing component

Design: Suggestion Engine

bonus content

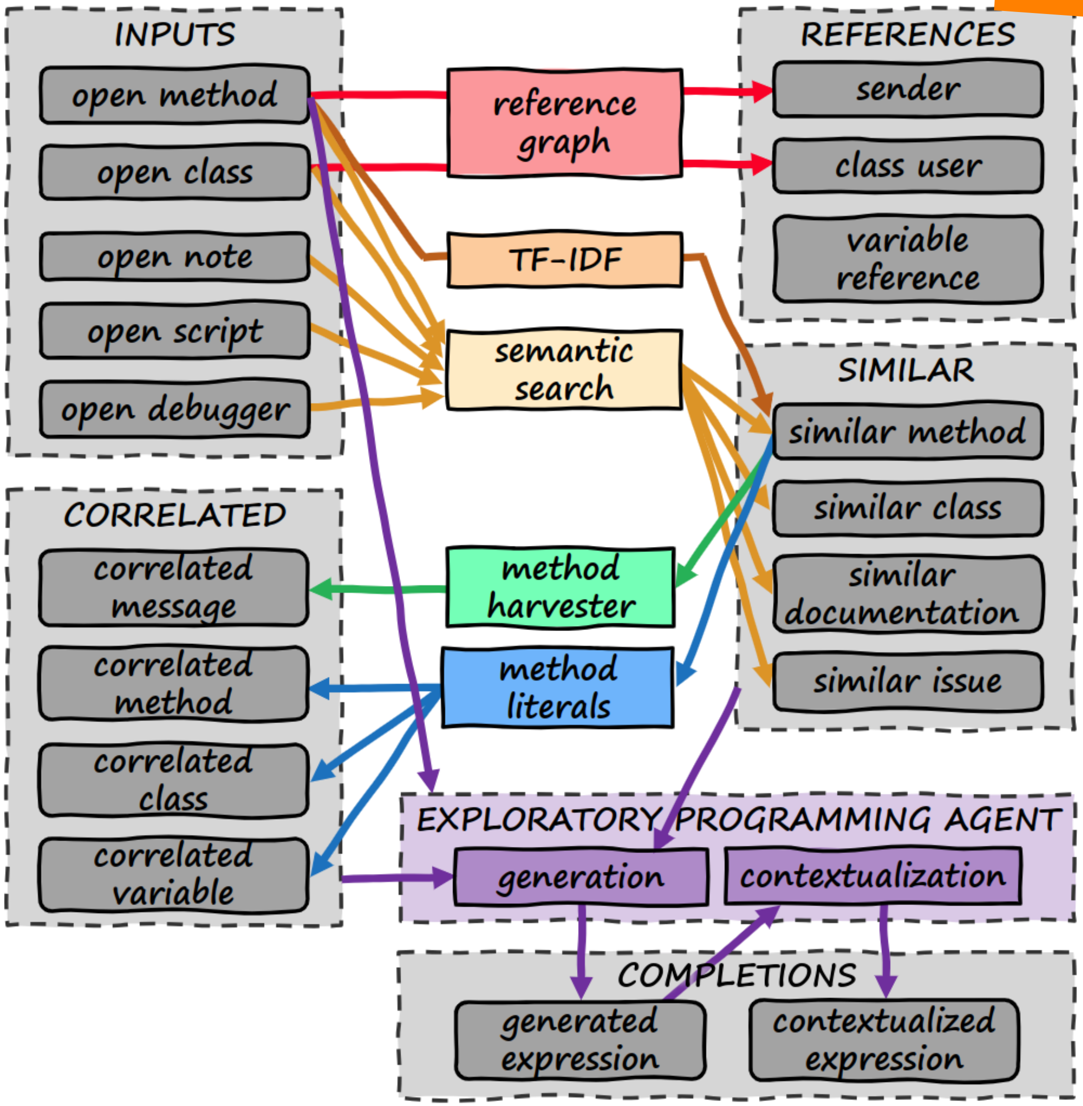




[HAY1985,BUS1996,TAL2013]

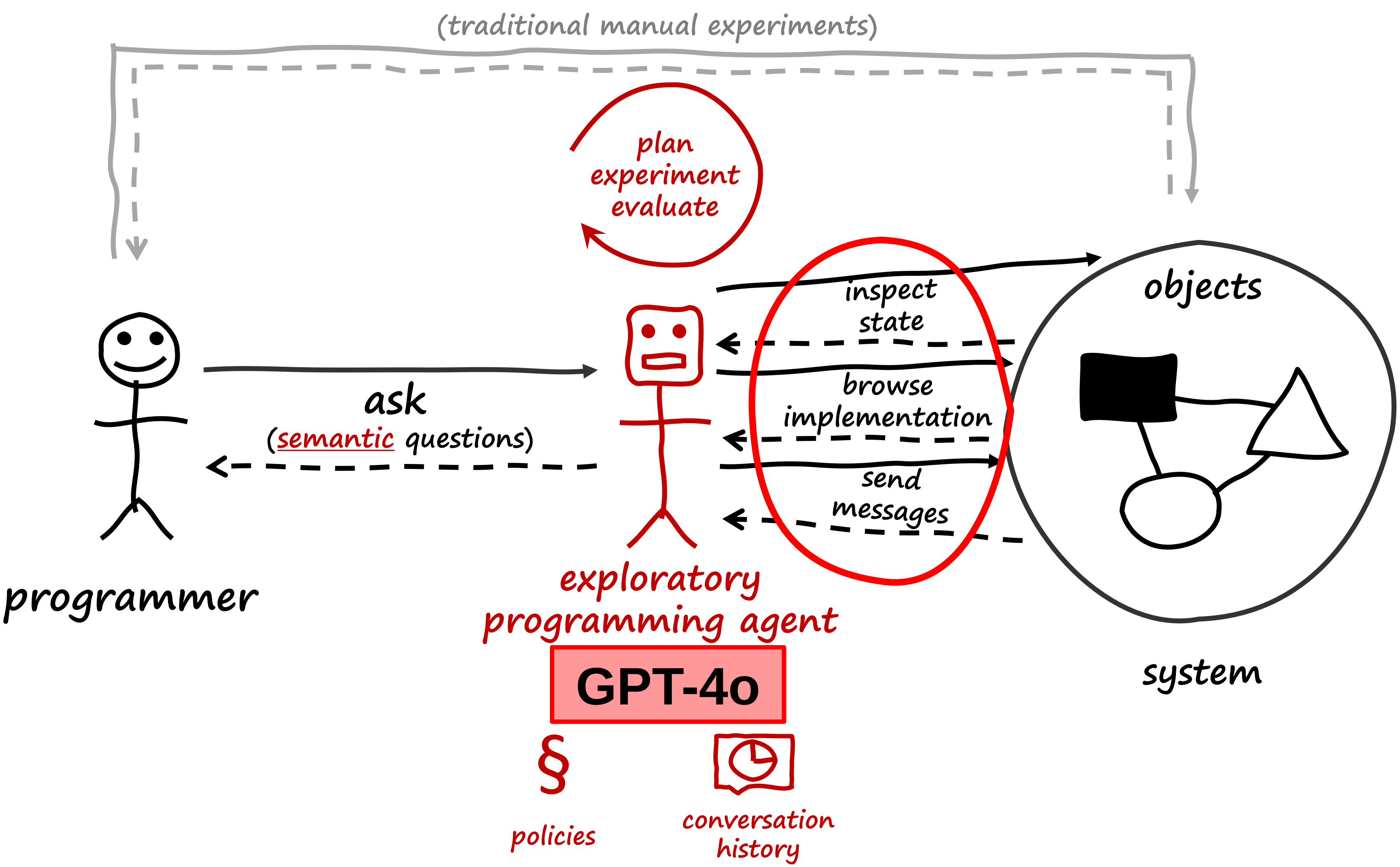
Design: Suggestion Engine

bonus content



[SAL1988,SU2009]

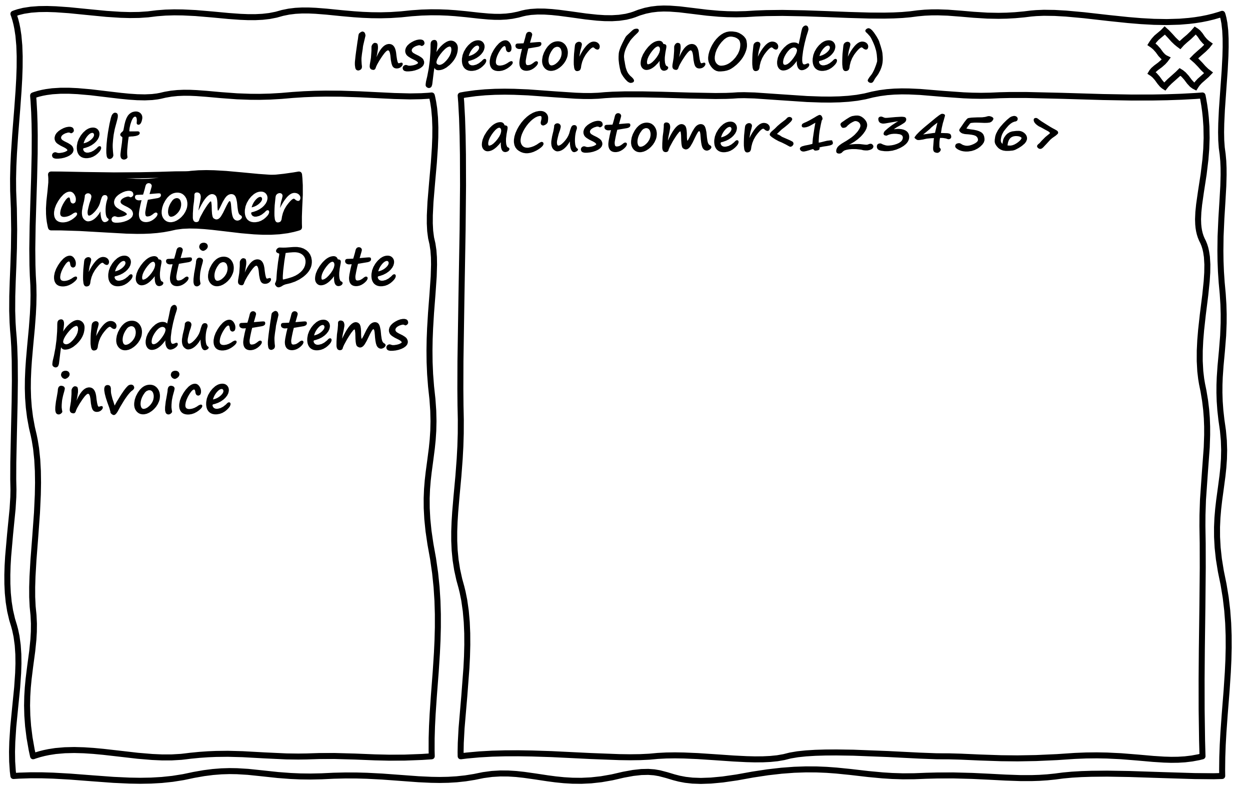
Design: Semantic Object Interfaces



[THI2024]

Design: Semantic Object Interfaces

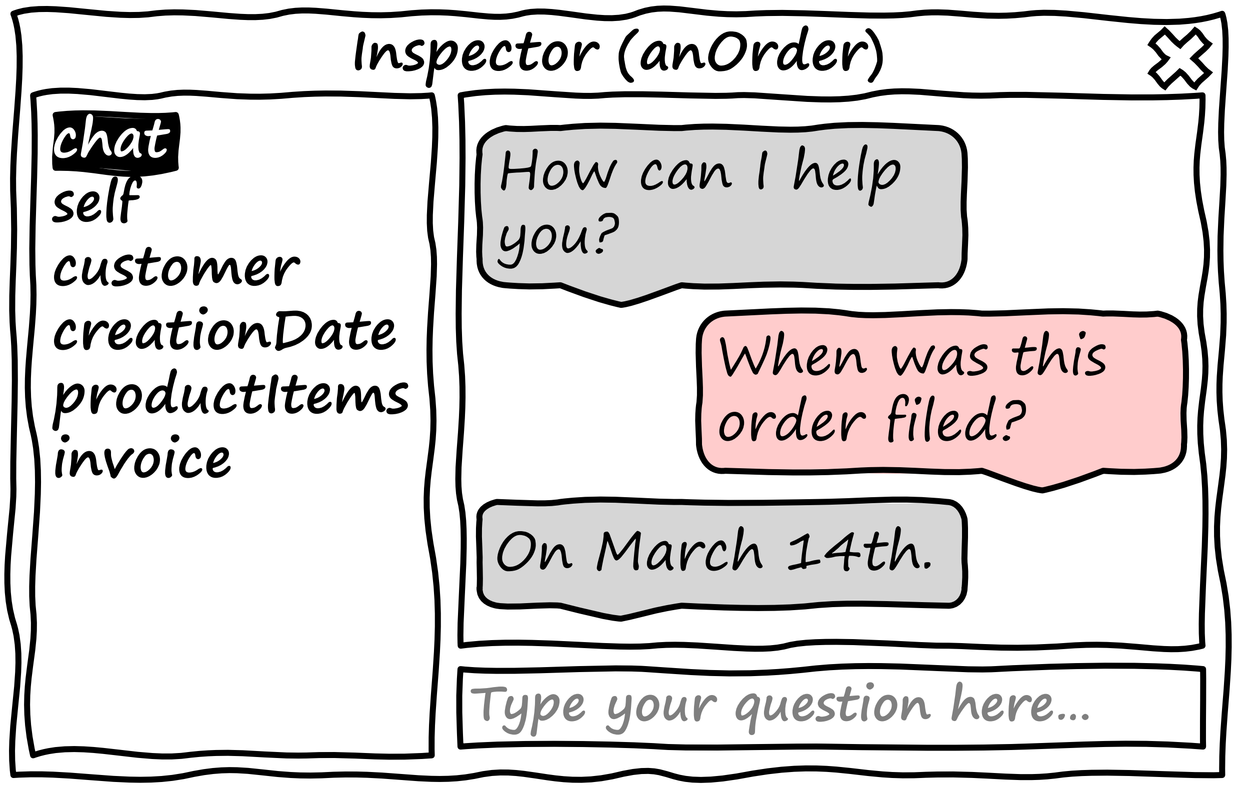
Conversation Mode for Object Inspection Tools



[GOL1984]

Design: Semantic Object Interfaces

Conversation Mode for Object Inspection Tools



Design: Semantic Object Interfaces

Semantic Messaging for Scripts

- Traditional scripting:
  - `aProduct customer lastName.`
  - `(aProduct orderItems detectMax: #quantity) product.`
- Scripting with semantic messages:
  - `aProduct orderItems mostOftenBoughtOne.`
  - `aProduct mostPopularArticle.`
  - `aProduct numberOfSalesTo: aCustomer.`
  - `aProduct countSalesFrom: '2023Q3' to: '2023Q4'.`

[SAN1988]

Implementation: Exploratory Programming Agent

Implementing Policies through Prompts

bonus content

HEADER

Exploratory programming agent

System: You are an exploratory programming agent... — *identity*
System: You can call the following functions... — *interface description*
System: To solve a task, you should... — *rules and traits for problem solving*

Conversation mode (optional)

System: You are an object... — *object identity*
System: Keep your answers brief... — *output format*

Semantic messaging (optional)

System: You must call the `evalAndReturn` function...
System: Format the return value as...
— *output format*

Bootstrapping the exploration

System: This object represents... — *hardcoded semantic context*
Assistant: To understand this object, I will first... — *zero-shot chain-of-thought*
Assistant: `eval("self printString")`
Result: `an Object(12345)`
Assistant: `eval("self allInstVarNames")`
Result: `#('foo' 'bar')`
— *initial object context*

BODY

User: What does this object...? — *user question*

[LEW2020,BRO2022]

Implementation: Exploratory Programming Agent

System Interfaces for Experiments

bonus content

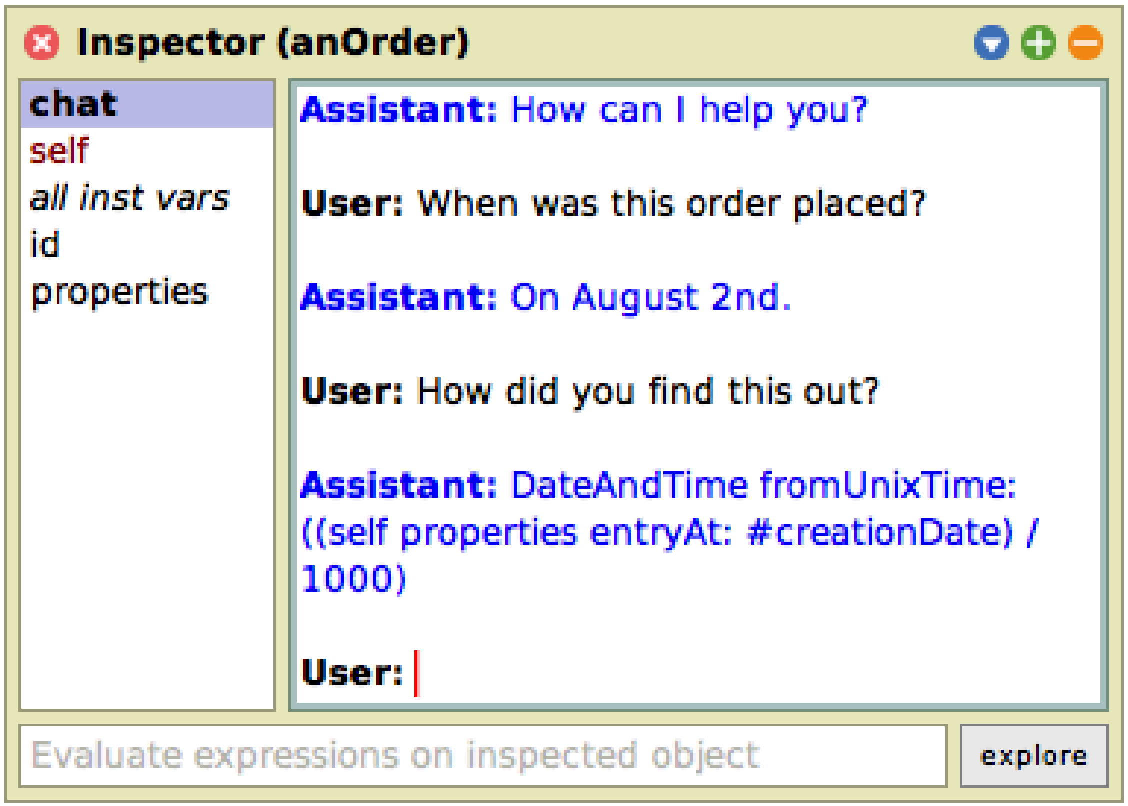
Function	Description
`eval(expression)` *Example:* `eval("self customer")`	Evaluate a Smalltalk expression in the context of the explored object and return the result or error. Can be executed in isolation.
`evalAndReturn(expression)`	Evaluate a Smalltalk expression in the context of the explored object and pass back the result to the sender of the original semantic message. Only available if the agent was invoked through a semantic message.
`browsePackage(packageName)`	Return a hierarchical list of classes within a package.
`browseClass(className)`	Enumerate all methods defined on a class or one of its superclasses or their metaclasses (for static methods), grouped by the defining class and the method category (protocol) within the class organization.
`browseMethod(className, selector)`	Retrieve the source code of a method defined in a class.

`browseSenders(selector[, query])`
Examples:
`browseSender("printOn:")`
`browseSender("printOn:", "date yyy-mm-dd")`
	Search the system for all methods that send messages with the name of a selector and return a subset.

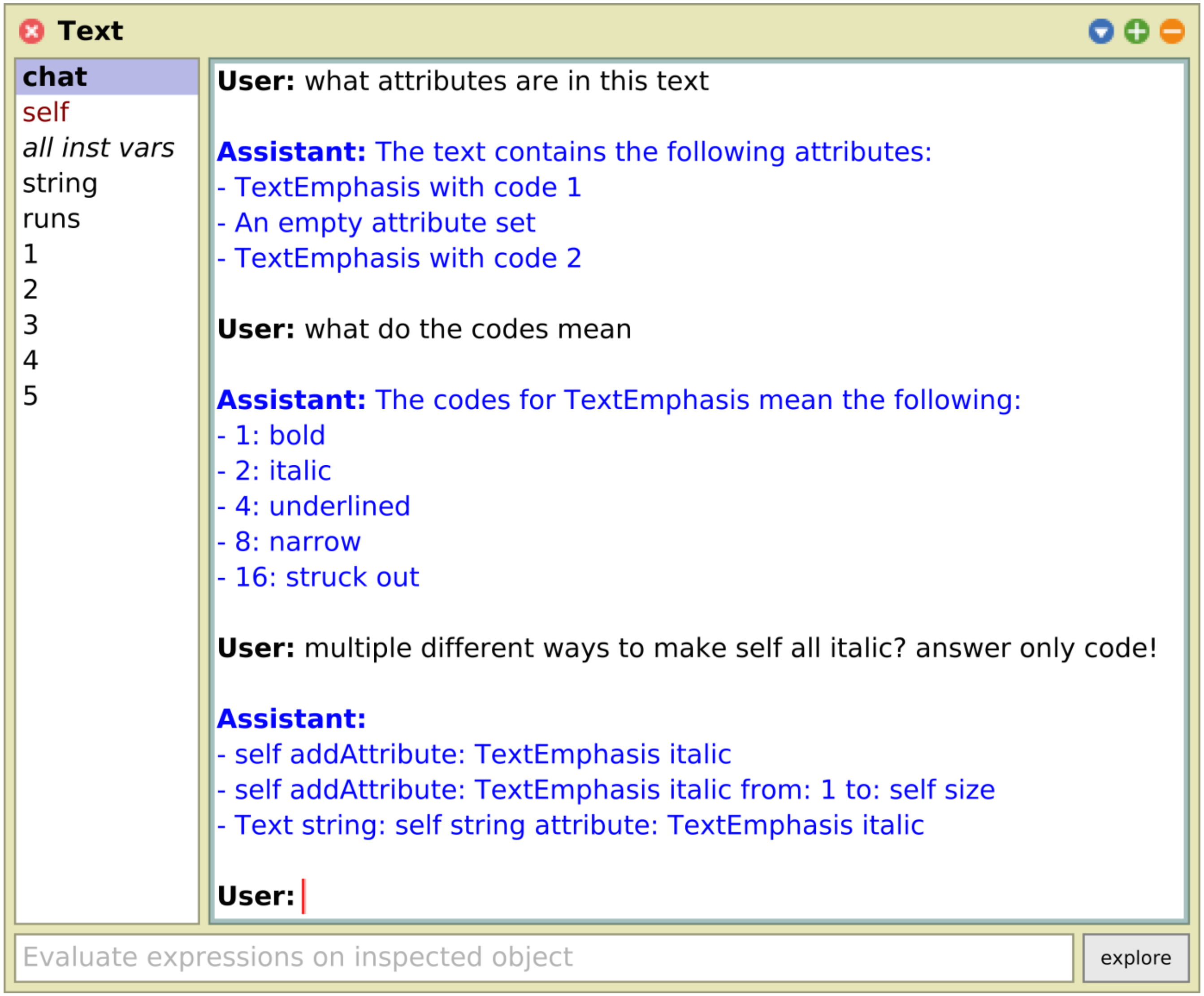
[MIA2023]

Demo

Demo: Order

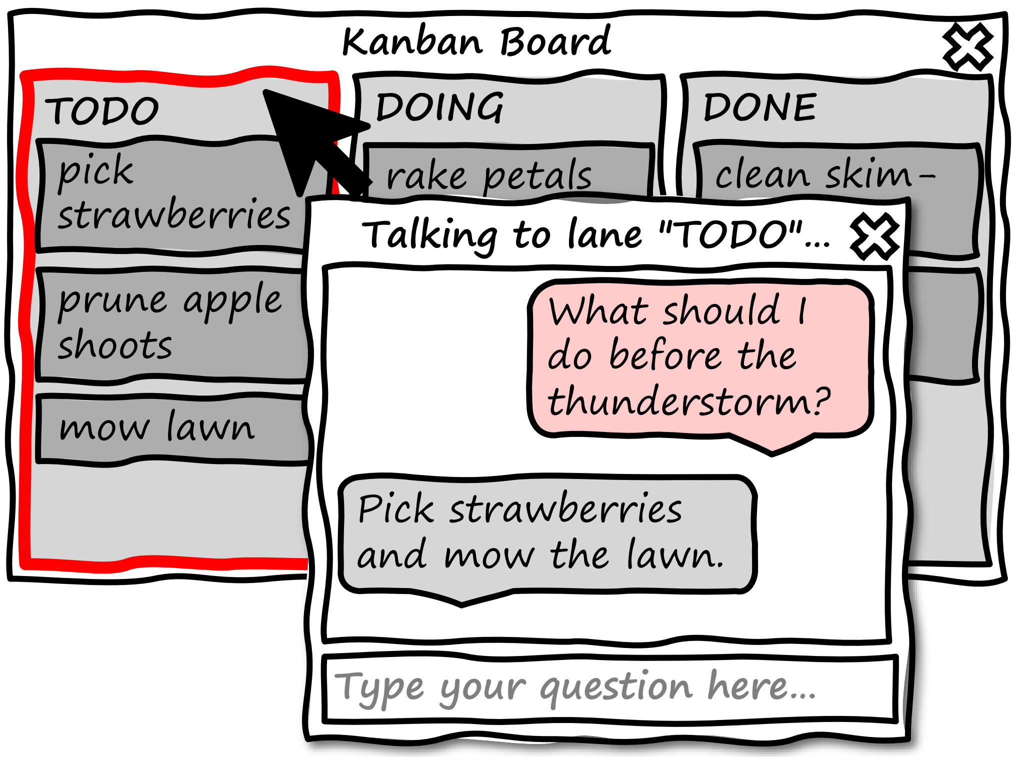

Inspector (anOrder)
chat
self
all inst vars
id
properties
Assistant: How can I help you?
User: When was this order placed?
Assistant: On August 2nd.
User: How did you find this out?
Assistant: DateAndTime fromUnixTime: ((self properties entryAt: #creationDate) / 1000)
User:
Evaluate expressions on inspected object
explore

Demo


Text
chat
self
all inst vars
string
runs
1
2
3
4
5
User: what attributes are in this text
Assistant: The text contains the following attributes:
- TextEmphasis with code 1
- An empty attribute set
- TextEmphasis with code 2
User: what do the codes mean
Assistant: The codes for TextEmphasis mean the following:
- 1: bold
- 2: italic
- 4: underlined
- 8: narrow
- 16: struck out
User: multiple different ways to make self all italic? answer only code!
Assistant:
- self addAttribute: TextEmphasis italic
- self addAttribute: TextEmphasis italic from: 1 to: self size
- Text string: self string attribute: TextEmphasis italic
User:
Evaluate expressions on inspected object
explore

Building a Semantic Toolset

- Idea: Allow users of object-oriented user interfaces to talk to domain objects on their screen



[PAW2001]

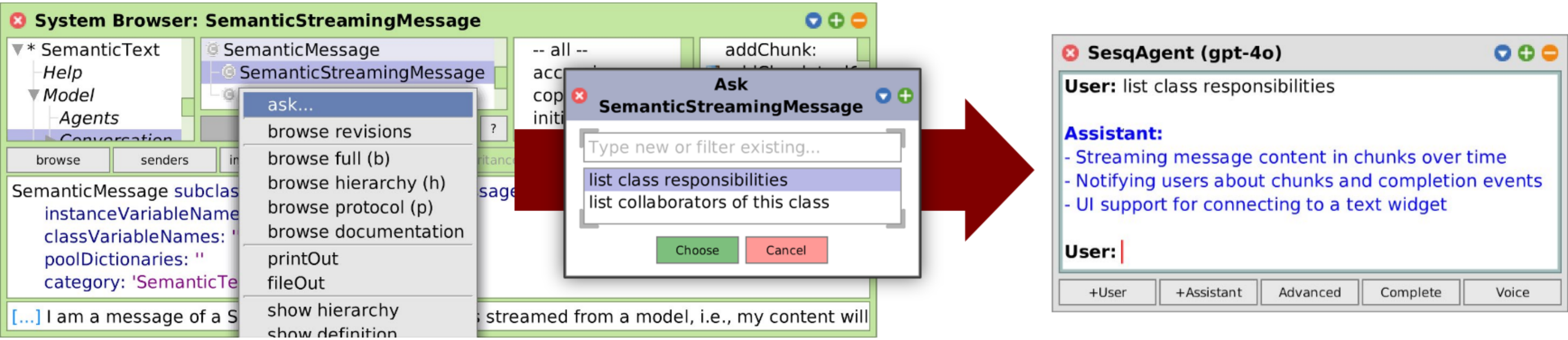
Building a Semantic Toolset

- Idea: Allow users of object-oriented user interfaces to talk to domain objects on their screen
- Many exploratory programming tools employ object-oriented interfaces:
  - Structural navigation tools (such as Smalltalk code browsers)
  - Projectional editors (based on AST)
  - Symbolic debuggers (based on process/call stack)
  - Profilers (based on trace)
  - ...

Building a Semantic Toolset: Browser

bonus content

- In a semantic code browser, programmers can engage in natural-language conversations with classes to explore them, e.g., by asking for their responsibilities or collaborators.

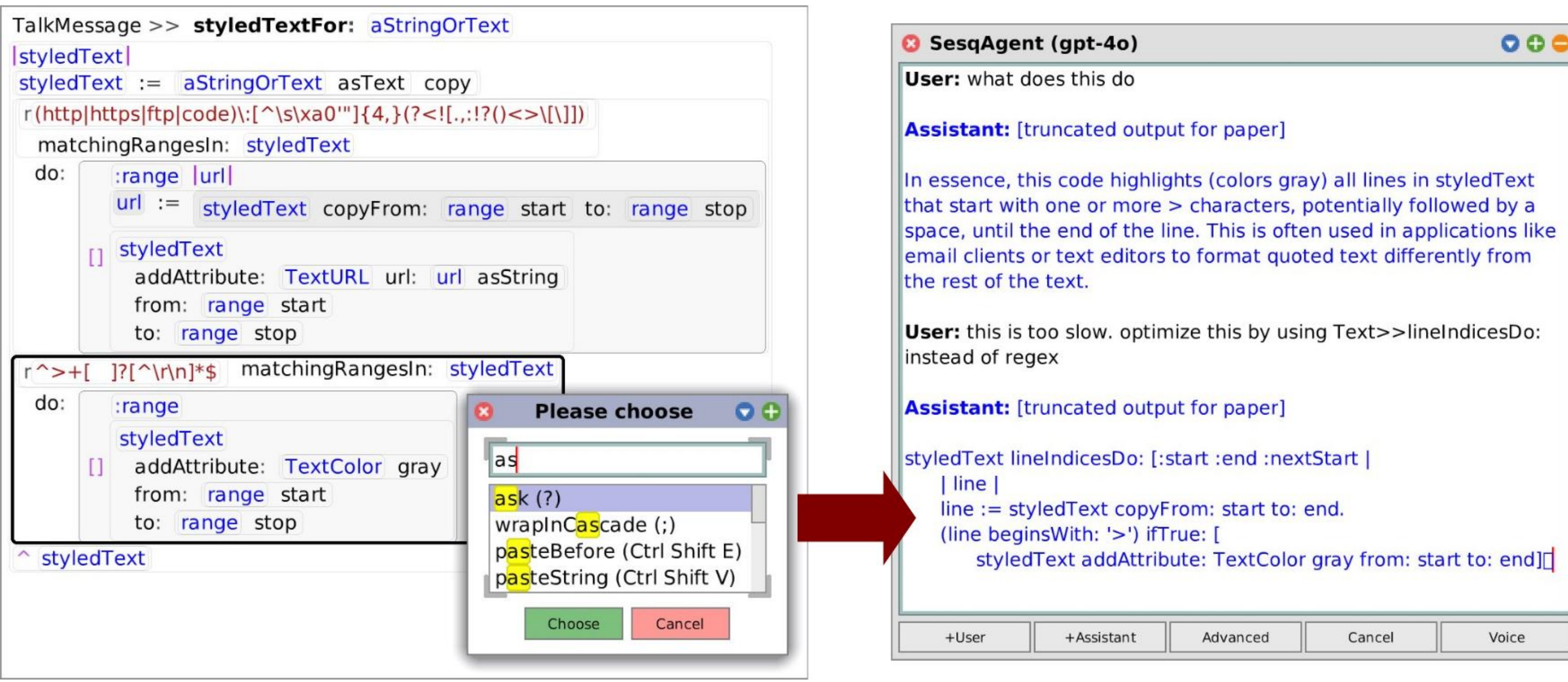


[Squeak]

Building a Semantic Toolset: Editor

bonus content

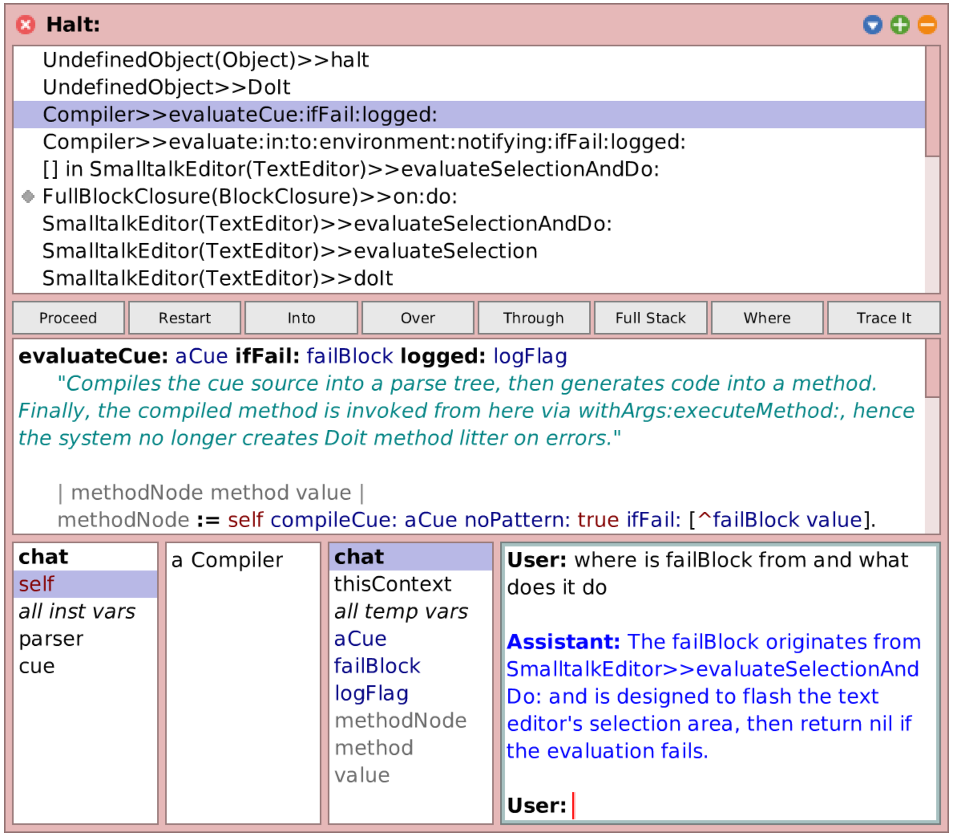
- In a semantic projectional editor (here: Sandblocks [BEC2020]), programmers can chat with single code blocks to explain, refactor, or execute them.



Building a Semantic Toolset: Debugger

bonus content

- In a semantic debugger, programmers can ask for the origin and meaning of values on the program stack.



[Squeak]

Discussion

How can we augment
the exploratory programming workflow
with semantic technologies?

- We integrate semantic interfaces into exploratory programming systems
  → **How capable are semantic technologies for our approach?**
- Exploratory programmers delegate work to intelligent agents and collaborate with semantic technologies
  → **How do semantic interfaces affect the exploratory programming experience?**
- This augments and streamlines their exploratory programming workflow

Discussion: Semantic Technologies

Semantic Retrieval

bonus content

Text embeddings

+
- Discover similar implementations
- Read relevant comments
- Spot duplications

−
- Limited conceptual comparison

→ Train custom model?

Term-based search

+
- Discover similar artifacts from prefixes

−
- False negatives from homonyms and synonyms
- False positives from polysemes
- Imprecise for artifacts with few terms

→ Use term-wise embeddings?

Ranking

+
- Combination of different strategies and objectives

−
- Sensitive to heterogeneity between different relevance scores

→ Normalize scores?

Discussion: Semantic Technologies

Capability of LLMs

- Problem solving:

Errors
Hallucinations, incorrect reasoning, invalid code

Failures
Insufficient answers, endless trial & error, refused tasks

Train specific abilities?
- Proficiency with Squeak/Smalltalk language + frameworks
- Exploratory practice

- Performance:

	Response times	Monetary cost	
Simple tasks	2 s – 4 s	$0.01 – $0.10	$1 – $60 per hour? [KUB2018]
Complex tasks	5 s – 15 s	$0.1 – $2	

Fine-tuned or small language models? [MAG2023]
Optimize prompts?

Discussion: Semantic Technologies

Performance (2024-09-30)

bonus content

Semantic interface	Response time	Monetary cost per query	Monetary cost per hour
Semantic suggestions	0.5 s – 0.9 s	\$0.0001 – \$0.001	\$0.08
Semantic completions			
Stage-1 generations	10 s – 15 s	\$0.15	\$18
Stage-2 generations	1.1 s – 1.3 s	\$0.0005	\$1.8
Total	11 s – 16.3 s		\$20
Semantic conversations			
Simple to moderate tasks	2 s – 4 s	\$0.1 – \$0.5	\$1 – \$60
Complex tasks	5 s – 10 s	\$0.5 – \$5	\$5 – \$30

- Memory consumption of embeddings: ~100 MB
- Ethical and environmental concerns

Discussion: Exploratory Programming Experience

Research process support

- Higher level of abstraction
  → Fewer interruptions [CSI2008]
- Tunnel vision: missed serendipitous discoveries

Natural language interfaces

- More intuitive/closer to mental model
  → Reduced gulf of execution/evaluation [NOR1986]
- Avoidance of explication

Delegation of control

- Limited trust

→ Need to improve explananation of semantic tools [CHEF2021]

Level of support

- Automation: more conceptual support, explicit invocation, separate interface, leaky abstraction [SPO2004]
- Augmentation: better integration with existing workflow

Related Work

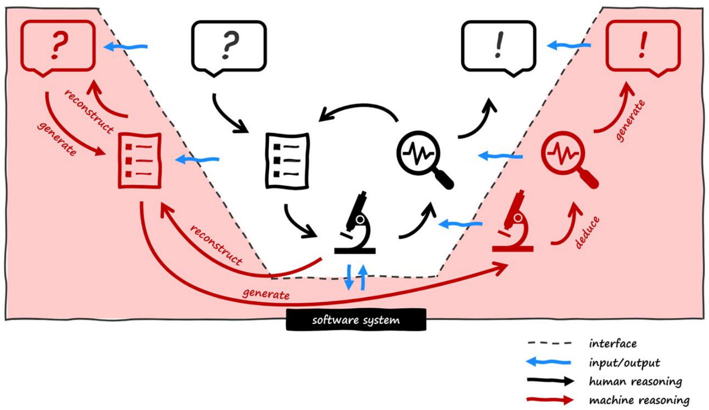
- Pair programming workflow with driver and navigator
- Suggestion tools
  - Traditional code completions
    - Microsoft IntelliSense, OCompletion [ROB2008], …
    - Palettes: Etoys, Scratch [RES2009], …
  - LLM code completions and refactorings: GitHub Copilot [BAR2023], …
- High-level programming interfaces
  - Question-based debugging: Whyline [KO2004], ChatDBG [LEV2024]
  - Conversational agents: GitHub Copilot Chat, ...
  - Natural-language programming: Navā [SAM2014], GPTScript, AIOS [MEI2024]

Future Work

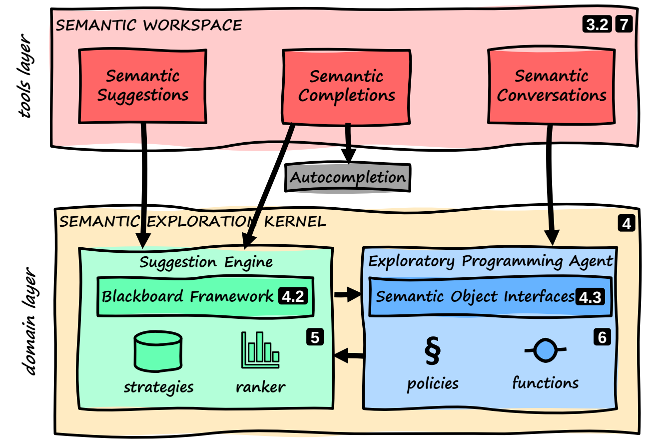
- How far can we improve the capability of semantic tools with existing language models?
  - Fine-tune, optimize, and configure language models
- How can we choose or combine augmentation vs. automation tools for an optimal programming experience?
  - Conduct qualitative or comparative user study
- How can we keep programmers in the loop when automating the research process?
  - Enhance collaboration between programmers and agents

Contributions

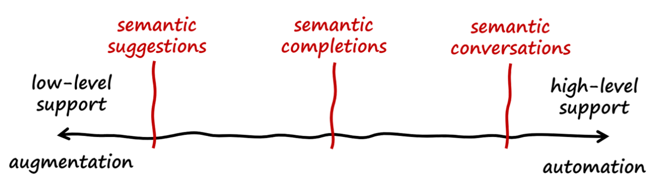
Model: Augmented exploratory programming workflow



Implementation: Semantic Exploration Kernel



Concept: Semantic Workspace



Prototype: Semantic Object Interfaces

The Thesis

- Christoph Thiede. 2024. *The Semantic Workspace: Augmenting Exploratory Programming with Integrated Generative AI Tools.* Master's Thesis. Hasso Plattner Institute, 161 pages.
- https://github.com/LinqLover/semexp-thesis/releases/download/submission/semexp-thesis.pdf

Acknowledgments

- Advisors & coauthors: Marcel Taeumel, Lukas Böhme, Robert Hirschfeld
- Machine learning education: Toni Mattis
- Three anonymous reviewers of the Onward! paper

- **Thank you for the beautiful time!**

Publications

- **Christoph Thiede, Marcel Taeumel, Lukas Böhme, and Robert Hirschfeld. Talking to Objects in Natural Language: Toward Semantic Tools for Exploratory Programming. In *Proceedings of the 2024 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! ’24)*, October 20–25, 2024, Pascadena, California. ACM, 17 pages.**
- Christoph Thiede, Willy Scheibel, and Jürgen Döllner. Bringing Objects to Life: Supporting Program Comprehension through Animated 2.5D Object Maps from Program Traces. In *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (IVAPP ‘24).* Volume 1: GRAPP, HUCAPP and IVAPP. INSTICC, Feb. 2024, Rome, Italy. SciTePress, 9 pages.
- Christoph Thiede, Marcel Taeumel, and Robert Hirschfeld. Time-Awareness in Object Exploration Tools: Toward In Situ Omniscient Debugging. In *Proceedings of SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! ’23)*, Oct. 2023, Cascais, Portugal. ACM, 15 pages.
- Christoph Thiede, Marcel Taeumel, and Robert Hirschfeld. Object-Centric Time-Travel Debugging: Exploring Traces of Objects. In *Companion Proceedings of the 7th International Conference on the Art, Science, and Engineering of Programming (<Programming>)*, Mar. 2023, Tokyo, Japan. ACM, 7 pages.
- Christoph Thiede, Willy Scheibel, Daniel Limberger, and Jürgen Döllner. Augmenting Library Development by Mining Usage Data from Downstream Dependencies. In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE ’22).* INSTICC, 2022. SciTePress, 2022, 12 pages.

Talks

- Talking to Objects in Natural Language: Toward Semantic Tools for Exploratory Programming. SPLASH Coference, Onward! Track, 2024-10-25, Pascadena, California, 20 minutes.
- SemanticText: Improving Exploratory Programming in Squeak with Generative AI. UKSTUG Meeting, 2024-11-27, 90 minutes.

Try It Out!

- https://github.com/hpi-swa-lab/SemanticSqueak
- Further reading:
  - https://github.com/hpi-swa-lab/Squeak-SemanticText
  - [squeak-dev] [ANN] Exploratory Programming - Talking to Objects in Natural Language
  - [squeak-dev] [ANN] SemanticText: ChatGPT, embedding search, and retrieval-augmented generation for Squeak
  - Transcript: Talking to Objects in Natural Language: Toward Semantic Tools for Exploratory Programming.

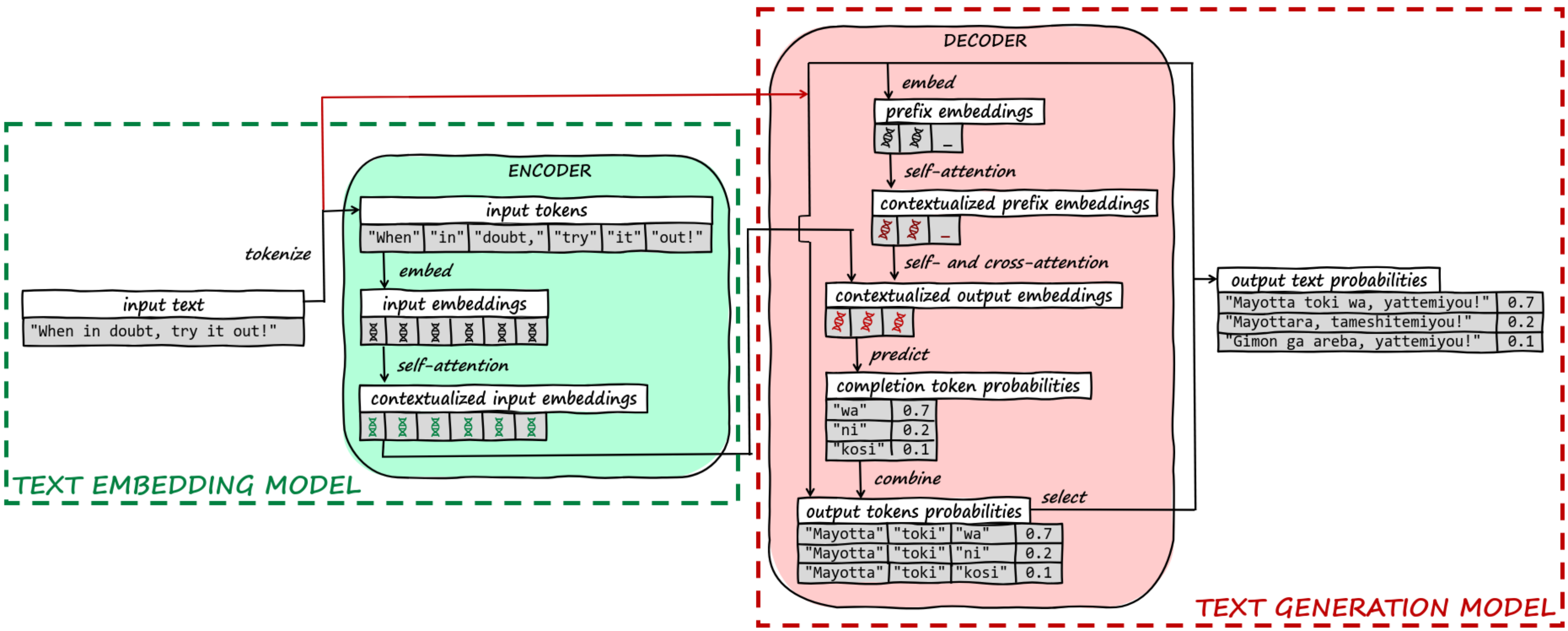
Bibliography (Programming & SE)

- [BAR2023] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proceedings of the ACM on Programming Languages 7*, OOPSLA1, Article 78 (4 2023), 27 pages. https://doi.org/10.1145/3586030
- [BEC2020] Tom Beckmann, Stefan Ramson, Patrick Rein, and Robert Hirschfeld. 2020. Visual Design for a Tree-Oriented Projectional Editor. In *Companion Proceedings of the 4th International Conference on the Art, Science, and Engineering of Programming* (Porto, Portugal) *(<Programming> '20 Companion)*. ACM, 113–119. https://doi.org/10.1145/3397537.3397560
- [BUS1996] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 1996. *Pattern-Oriented Software Architecture Volume.* Volume 1: A System of Patterns. Wiley. ISBN: 978-0-471-95869-7.
- [CHI2015] Andrei Chiş, Oscar Nierstrasz, Aliaksei Syrel, and Tudor Gîrba. 2015. The Moldable Inspector. In: *International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Pittsburgh, PA, USA) *(Onward! 2015)*. ACM, 44–60. https://doi.org/10.1145/2814228.2814234
- [CSI2008] Mihaly Csikszentmihalyi. 2008. *Flow: The Psychology of Optimal Experience* (1 ed.). Harper Perennial, New York. ISBN: 978-0-06-133920-2.
- [GOL1984] Adele Goldberg. 1984. *Smalltalk-80: The Interactive Programming Environment.* Addison-Wesley. ISBN: 978-0-201-11372-3.
- [KO2004] Amy J. Ko and Brad A. Myers. 2004. Designing The Whyline: A Debugging Interface for Asking Questions about Program Behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria) *(CHI '04)*. ACM 151–158. https://doi.org/10.1145/985692.985712
- [LEV2024] Kyla Levin, Nicolas van Kempen, Emery D. Berger, and Stephen N. Freund. 2024. *ChatDBG: An AI-Powered Debugging Assistant.* arXiv:2403.16354 [cs.SE]
- [MEI2024] Kai Mei, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. 2024. *AIOS: LLM Agent Operating System.* arXiv: 2403.16971 [cs.OS]
- [NOR1986] Donald A. Norman. 1986. *Cognitive Engineering.* Lawrence Erlbaum, Hillsdale, NJ, USA, 31–61. ISBN: 978-0-367-80732-0.
- [PAW2001] Richard Pawson and Robert Matthews. 2001. Naked Objects: A Technique for Designing More Expressive Systems. In *SIGPLAN Not.* 36.12, 61–67. https://doi.org/10.1145/583960.583967
- [REI2019] Patrick Rein, Stefan Ramson, Jens Lincke, Robert Hirschfeld, and Tobias Pape. 2019. Exploratory and Live, Programming and Coding: A Literature Study Comparing Perspectives on Liveness. *The Art, Science, and Engineering of Programming* 3, 1 (07 2019), 33 pages. https://doi.org/10.22152/programming-journal.org/2019/3/1
- [RES2009] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. In *Communications of the ACM* 52.11 (Nov. 2009), 60–67. https://doi.org/10.1145/1592761.1592779
- [ROB2008] Romain Robbes and Michele Lanza. 2008. How Program History Can Improve Code Completion. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE '08)*. IEEE, 317–326. https://doi.org/10.1109/ASE.2008.42
- [SAM2014] Hesam Samimi, Chris Deaton, Yoshiki Ohshima, Alessandro Warth, and Todd Millstein. 2014. Call by Meaning. In *Proceedings of the 2014 International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (Portland, Oregon, USA) *(Onward! 2014)*. ACM, 11–28. https://doi.org/10.1145/2661136.2661152
- [SAN1988] David W. Sandberg. 1988. Smalltalk and Exploratory Programming. *SIGPLAN Not.* 23, 10 (1988), 85–92. https://doi.org/10.1145/51607.51614
- [SPO2004] Joel Spolsky. *Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity.* Berkeley, CA, USA: Apress, 2004. ISBN: 978-1-59059-389-9.
- [TAE2022] Marcel Taeumel, Jens Lincke, Patrick Rein, and Robert Hirschfeld. 2022. A Pattern Language of an Exploratory Programming Workspace. In *Design Thinking Research: Achieving Real Innovation,* Christoph Meinel and Larry Leifer (Eds.). Springer, Cham, 111–145. https://doi.org/10.1007/978-3-031-09297-8_7
- [TAL2013] Domenico Talia. 2013. Workflow Systems for Science: Concepts and Tools. In *ISRN Software Engineering 2013.1*, page 15. https://doi.org/10.1155/2013/404525
- [THI2024] Christoph Thiede, Marcel Taeumel, Lukas Böhme, and Robert Hirschfeld. 2024. Talking to Objects in Natural Language: Toward Semantic Tools for Exploratory Programming. In *Proceedings of SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Pasadena, CA, USA) *(Onward! '24)*. ACM, 17 pages. https://doi.org/10.1145/3689492.3690049
- [UNG1997] David Ungar, Henry Lieberman, and Christopher Fry. 1997. Debugging and the Experience of Immediacy. In *Communications of the ACM* 40, 4, 38–43. https://doi.org/10.1145/248448.248457

Bibliography (AI & Information Retrieval)

- [BRO2022] Tom B. Brown et al. 2020. *Language Models are Few-Shot Learners.* 2020. arXiv:2005.14165 [cs.CL]
- [CHEF2021] Hila Chefer, Shir Gur, and Lior Wolf. 2021. Generic Attention-Model Explainability for Interpreting Bi-Modal and Encoder-Decoder Transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV '21)*, pages 387–396. https://doi.org/10.1109/ICCV48922.2021.00045
- [CHEN2021] Mark Chen et al. 2021. *Evaluating Large Language Models Trained on Code.* arXiv:2107.03374 [cs.LG]
- [DEV2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (Minneapolis, Minnesota, USA), volume 1. Association for Computational Linguistics, pages 4171–4186. https://doi.org/10.18653/v1/N19-1423
- [HAY1985] Barbara Hayes-Roth. 1985. A Blackboard Architecture for Control. In *Artificial Intelligence* 26.3, pages 251–321. https://doi.org/10.1016/0004-3702(85)90063-3
- [KUB2018] Juraj Kubelka, Romain Robbes, and Alexandre Bergel. 2018. The Road to Live Programming: Insights from the Practice. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. ACM, 1090–1101. https://doi.org/10.1145/3180155.3180200
- [LEW2020] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih,Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems* (Vancouver, Canada) *(NIPS '20, Vol. 33)*. Curran, RedHook, NY, USA, Article 793, 16 pages. ISBN: 978-1-7138-2954-6. arXiv:2005.11401 [cs.CL]
- [MAG2023] Lucie Charlotte Magister et al. 2023. Teaching Small Language Models to Reason. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, Vol. 2: Short Papers. Association for Computational Linguistics, Toronto, Canada, 1773–1781. https://doi.org/10.18653/v1/2023.acl-short.1511
- [MIA2023] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. *Augmented Language Models: A Survey.* arXiv:2302.07842 [cs.CL]
- [MIK2013] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In: *1st International Conference on Learning Representations* (Scottsdale, Arizona) *(ICLR 2013)*, Workshop Track Proceedings. arXiv: 1301.3781 [cs.CL]
- [RAD2018] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. *Improving Language Understanding by Generative Pre-Training.* https://openai.com/research/language-unsupervised
- [SAL1988] Gerard Salton and Christopher Buckley. 1988. Term-Weighting Approaches in Automatic Text Retrieval. In *Information Processing & Management* 24.5, pages 513–523. https://doi.org/10.1016/0306-4573(88)90021-0
- [SU2023] Hongjin Su, Weijia Shi, Jungo Kasai, Yizhong Wang, Yushi Hu, Mari Ostendorf, Wen-tau Yih, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. 2023. *One Embedder, Any Task: Instruction-Finetuned Text Embeddings.* arXiv:2212.09741 [cs.CL]
- [WAY2023] Wayne Xin Zhao et al. 2023. *A Survey of Large Language Models.* 124 pages. arXiv:2303.18223 [cs.CL]

Background: Transformer Architecture



Design: Semantic Suggestions

Correlation Search

Example. A programmer is writing a script to create a red circle. Their incomplete draft looks like this:

```
circle := Circle new.
color := Color red.
```

Based on the used names, the suggestion engine identifies similar methods such as:

- High similarity (3 common terms):

```
circle := Circle new.
circle color: Color green.
circle border: #thick.
```

- Moderate similarity (2 common terms):

```
triangle := Triangle new.
triangle color: Color green.
triangle shadow: true.
```

- Low similarity (1 common term):

```
rectangle := Rectangle newSquare.
rectangle borderColor: Color blue.
```

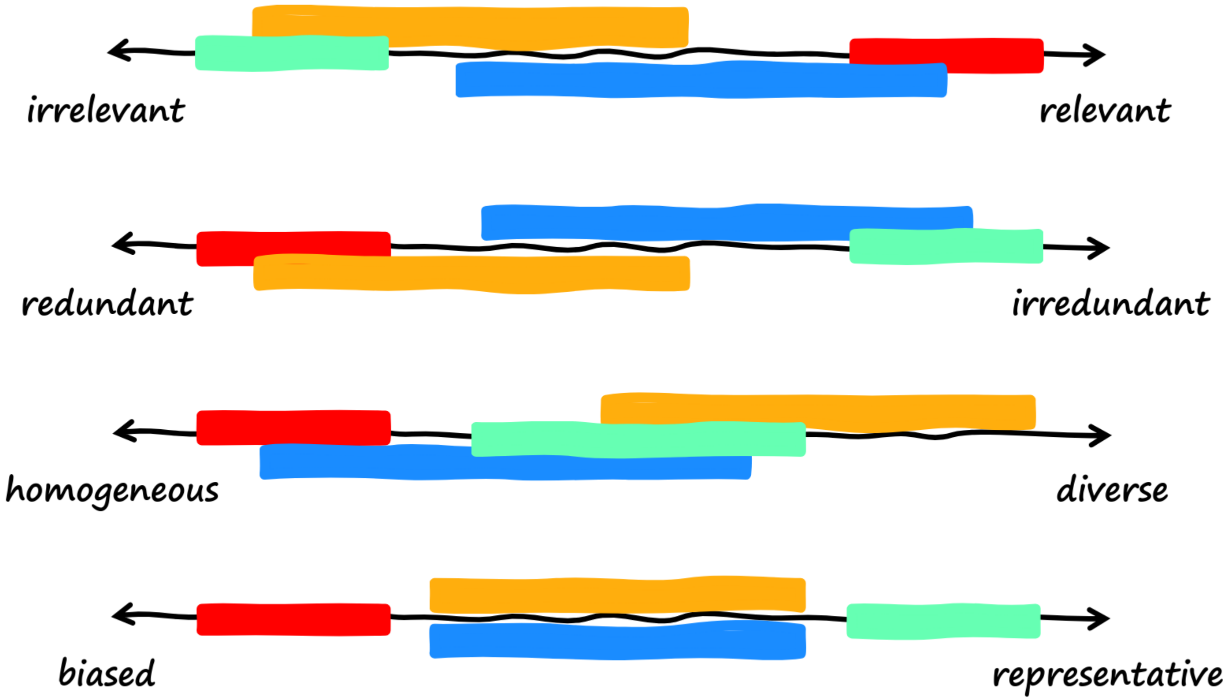
From these similar methods, the suggestion engine suggests the following most relevant new correlated artifacts:

- `#color:` (used in 1 highly similar and 1 moderately similar method)
- `#green` (used in 1 highly similar and 1 moderately similar method)
- `#border:` (used in 1 highly similar method)
- `#shadow:` (used in 1 moderately similar method)

Thus, the programmer can complete their script by choose from the most likely suggestions.

Implementation: Semantic Suggestions

Ranking



- Top-k selection
- Probabilistic sampling
  - $p_i = \frac{e^{\frac{R_i}{T}}}{\sum_j e^{\frac{R_j}{T}}}$ (R_i: relevance score, T: temperature)
- Clustering
- Probabilistic sampling from clusters

Implementation: Semantic Completions

Prompt	
Code completion agent	
System: You are a code completion agent...	• *identity*
System: You will complete a method...	• *task and rules*
System: Use the following information...	• *data description*
Example	
User: `DateAndTime»yyyymmddString` `^`	• *draft*

User: `self`: `2024-06-22T00:30:37.216061+02:00`
 `class`: `DateAndTime`
 `utcMicroseconds`: `1719009070988843`
 `...`
	• *receiver object state*

User: `Magnitude subclass: #DateAndTime...` `('accessing' getSeconds -> 1870 setSeconds:...`	• *receiver class: definition and protocols with preview results*

User: Related classes and methods:
 `ArrayedCollection subclass: #String...`
 example: `'hi'`
 `('accessing' byteAt: byteSize -> 2 ...`
	• *correlated classes: definition and protocols with preview results*

User: `SequenceableCollection»streamContents: blockWithArg`
 `^ self new...`
 `Text class»exampleWithNumber: x`
 `<primitive: "primitiveCopyBits"...`
	• *correlated methods: definition and implementation*

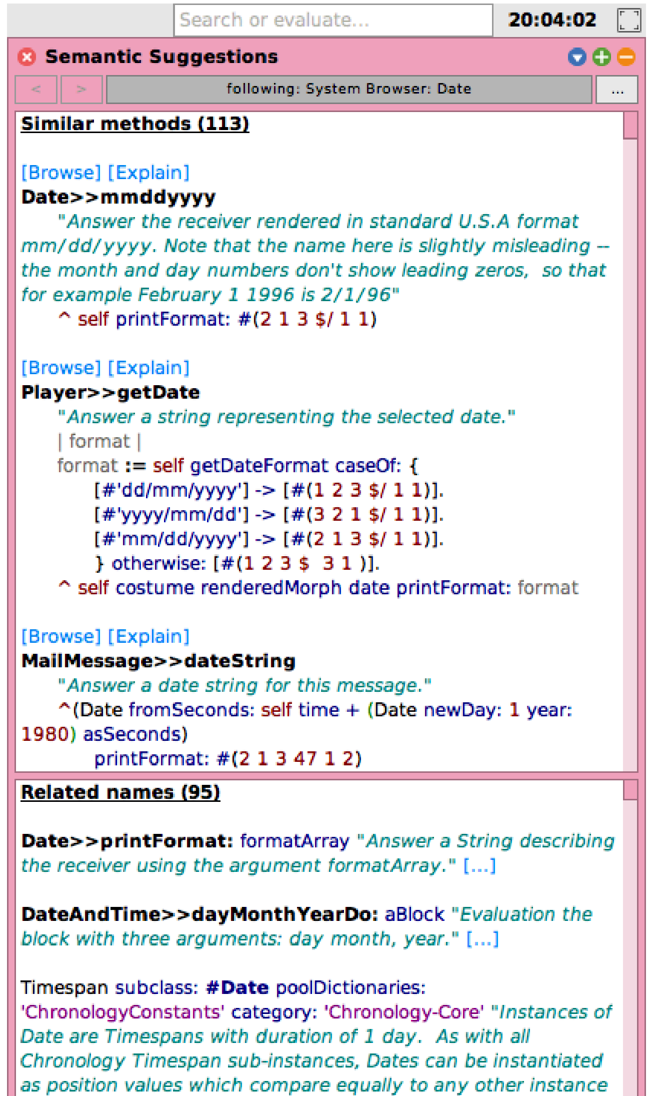
Assistant: `DateAndTime»yyyymmddString` `^ String streamContents: [:stream | self...`	• *completion*
Task	
User: ⟨*information about task in the same format as above*⟩	
System: Now complete this: **User:** ⟨*draft again*⟩	• *repetition of the task context*

stage 1

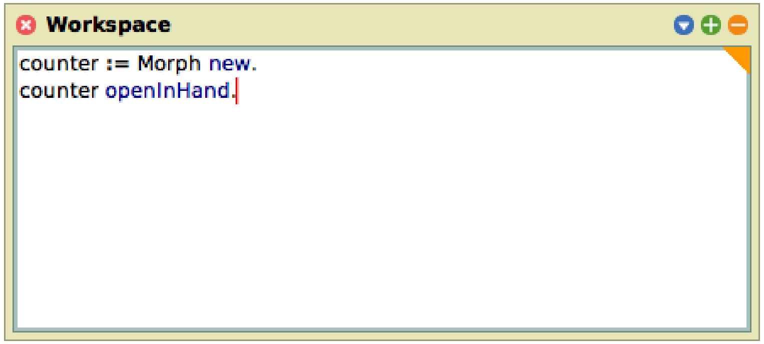
Prompt	
Task	
System: Use the following code snippet...	• *task and rules (brief)*
User: ⟨*stage-1 expression*⟩	• *previously generated code completion*
System: Now complete this: **User:** ⟨*updated draft*⟩	• *task context*

stage 2

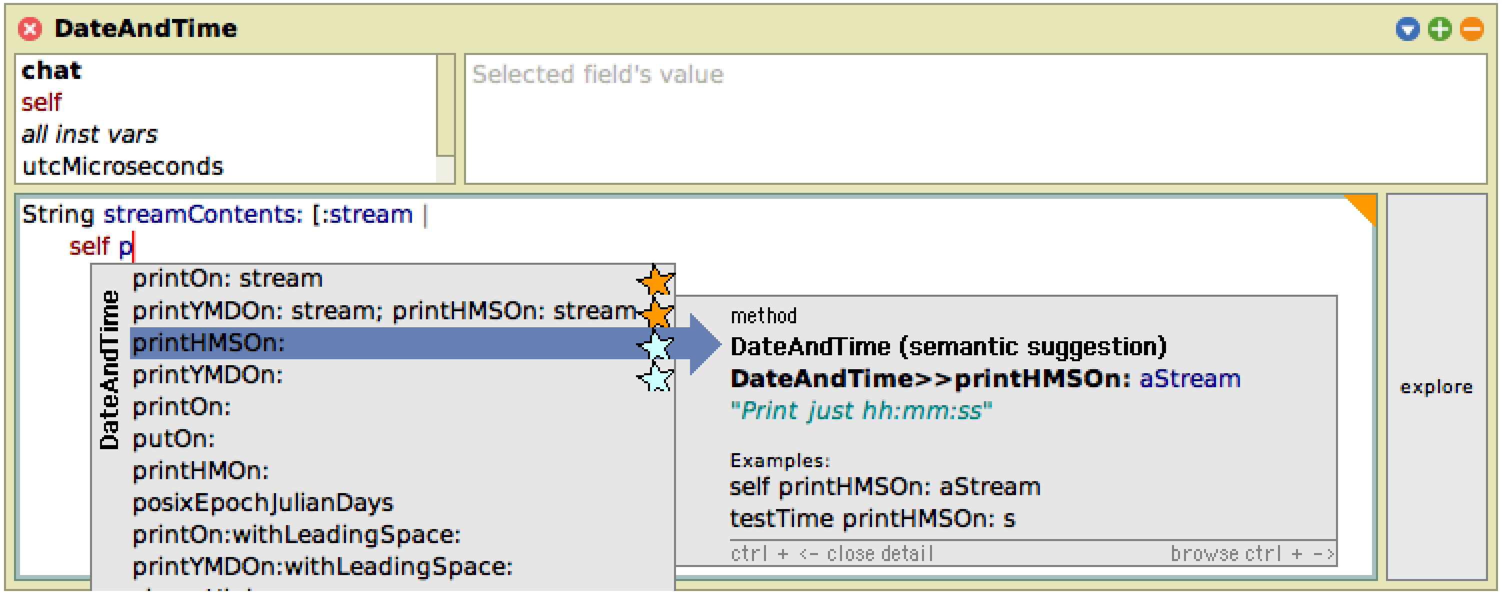
Demo: Semantic Suggestions



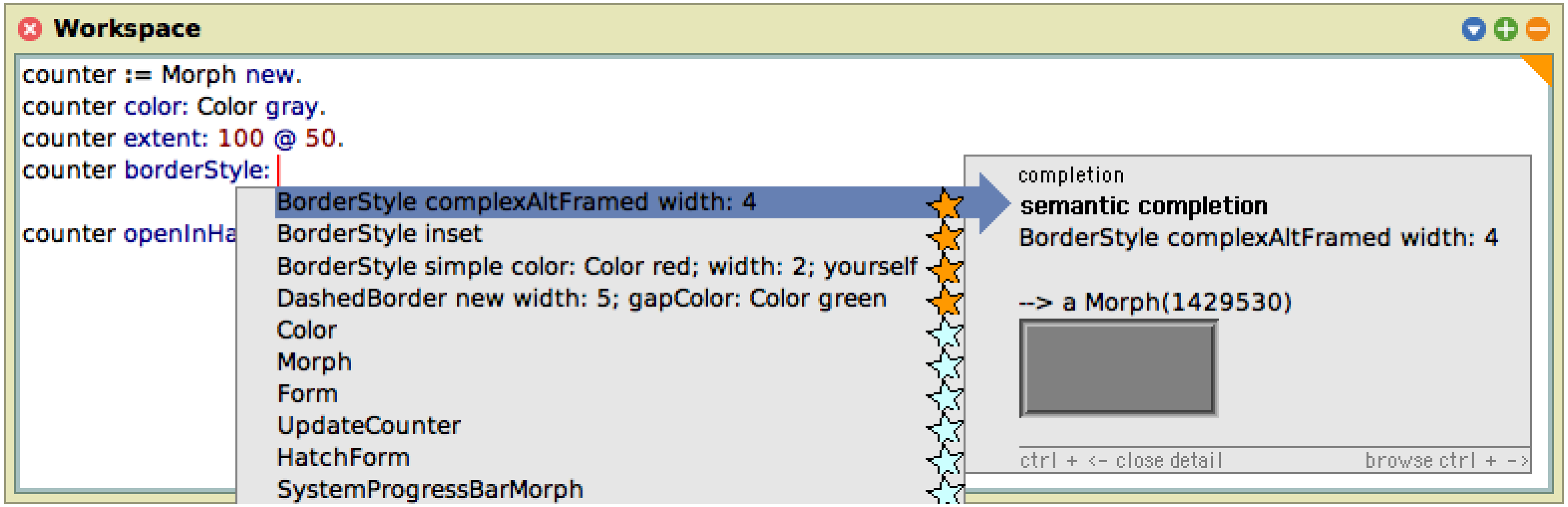
Demo: Semantic Suggestions



Demo: Semantic Completions



Demo: Semantic Completions



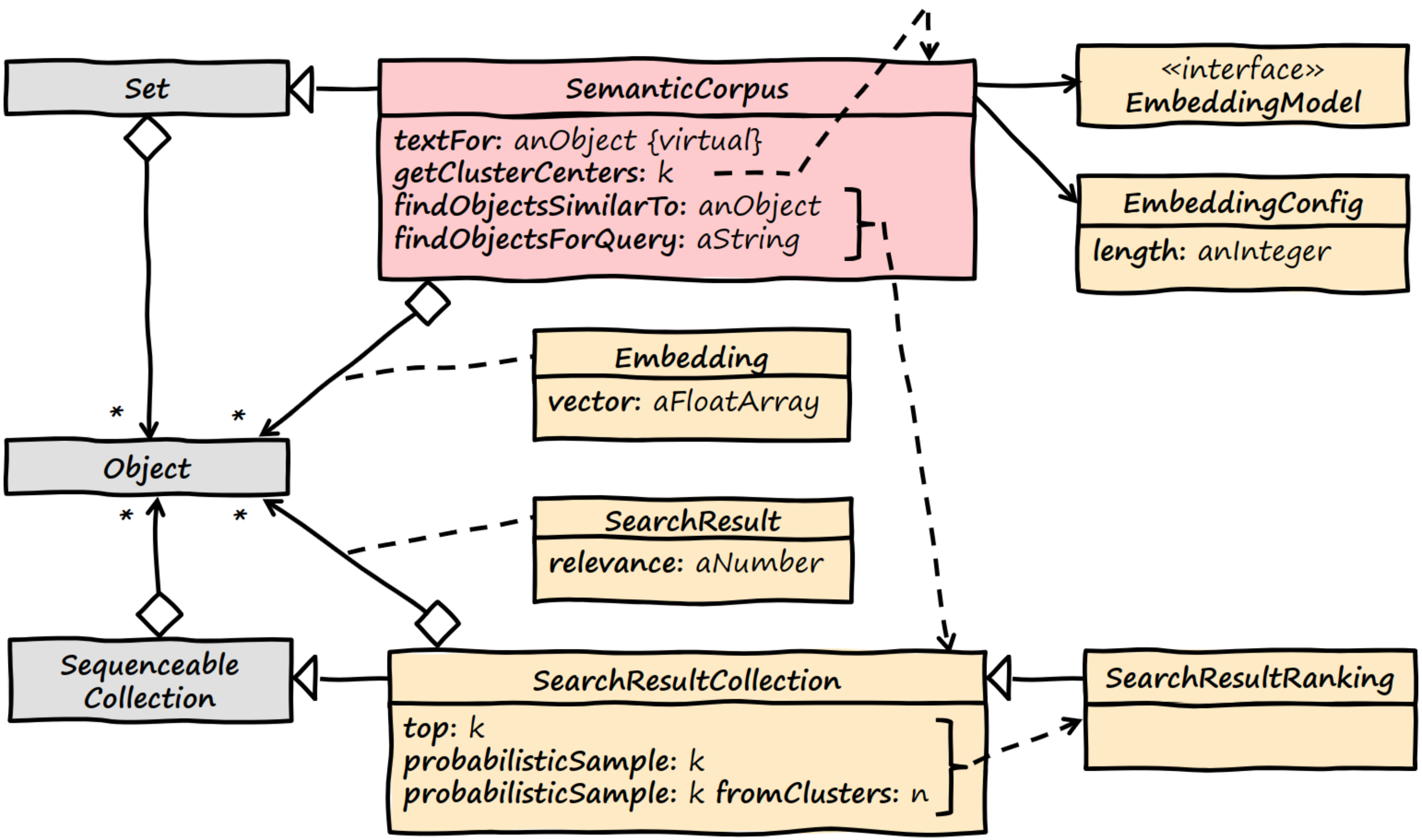
Discussion: Ethical Considerations of Language Models

- Concentration of economic and political power
- High energy intake (one question to agent: 0.05 kWh)
- High water consumption (one question to agent: 50-750 ml)
- Intellectual property of training data
- Working conditions of click workers for training data
- Poorly understood biases: safety, accessibility, decision making

HPI

Appendix: SemanticText

Domain Model: Semantic Retrieval



Appendix: SemanticText

Domain Model: Semantic Retrieval

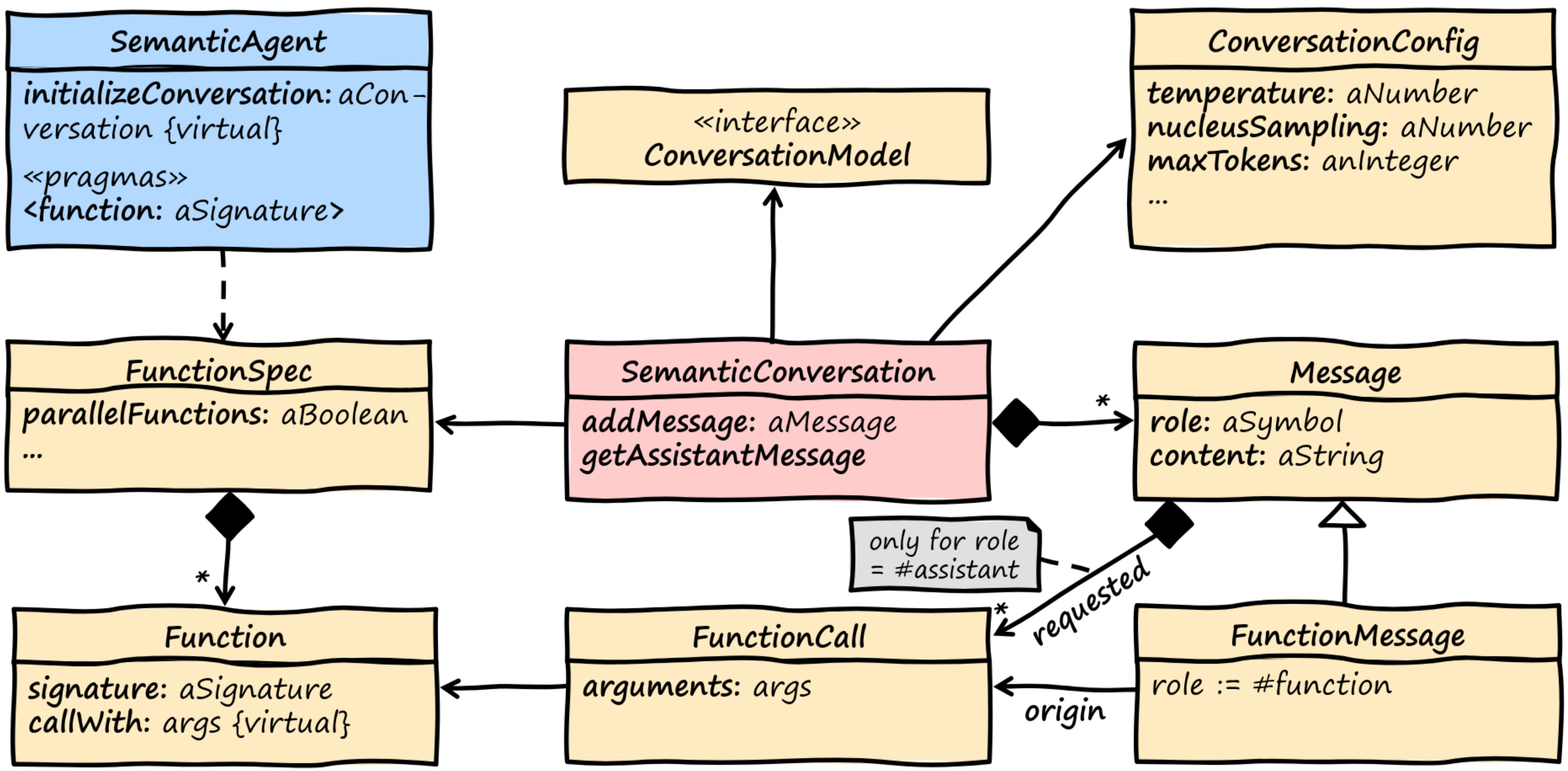
Example. A programmer wants to find classes in the system that implement means for semantic search. For this, they can create a semantic corpus of all classes based on their names and comments, perform a search, and rank the results:

```
corpus := self systemNavigation allClasses
    asSemanticCorpusWithTitle: #name
    content: #comment.
results := corpus findObjectsForQuery: 'semantic
search database'.
ranking := results top: 5.
ranking →a a SearchResultRanking(
#SemanticCorpus->0.533
#SemanticHelpSearchTopic->0.442
#SemanticText->0.385 #SemanticAgentParser->0.364
#SemanticMathAgent->0.338)
```

[a] We use the notation <expr> → <*result*> to indicate a *print-it* evaluation [67, p. 13].

Appendix: SemanticText

Domain Model: Conversations



Appendix: SemanticText

Domain Model: Conversations

Example. A programmer wants to create a chatbot that can retrieve the current time and date. For this, they define a conversation with an appropriate configuration for the LLM, define the necessary functions, and provide the question of the user:

```
SemanticConversation new
    withConfigDo: [:config |
        config temperature: 0.2];
    addFunction: #getTime action: [Time now];
    addUserMessage: 'What time is it?';
    getAssistantReply → 'The current time is
13:59.'
```

Appendix: SemanticText

Domain Model: Conversations

Example. A programmer wants to build a chatbot that can access the running Squeak image to assist the user. To achieve this, they create a subclass of `SemanticAgent`, initialize the conversation, and define an `#eval:` method:

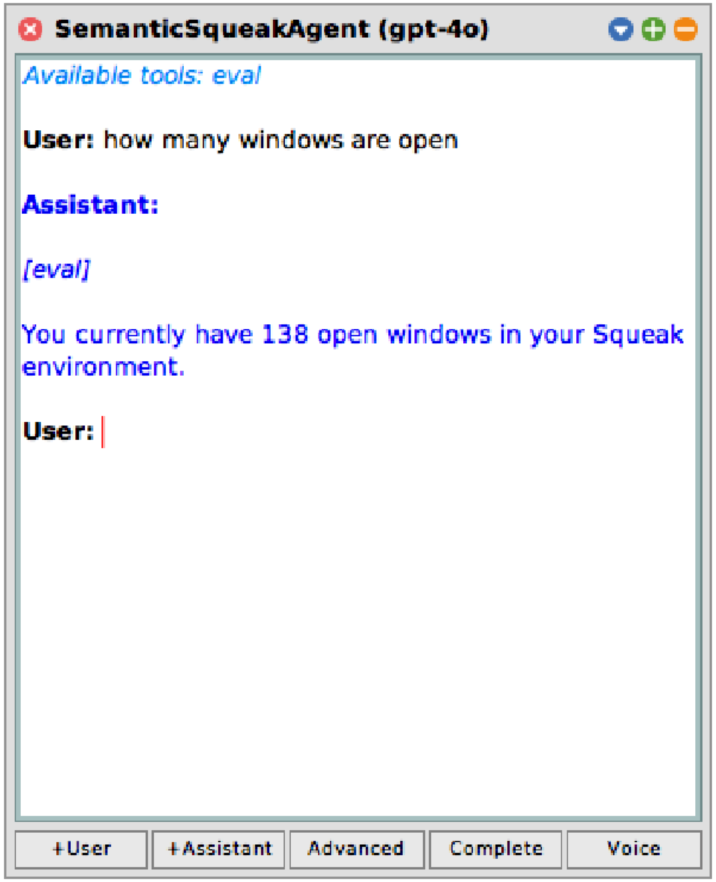
```
SemanticAgent subclass: #SemanticSqueakAgent
   instanceVariableNames: ''
   classVariableNames: ''
   poolDictionaries: ''
   category: 'SemanticText-Model-Agents'

SemanticSqueakAgent»initializeConversation:
aConversation
   super initializeConversation: aConversation.
   aConversation addSystemMessage: 'You are a
Squeak/Smalltalk assistant.'.

SemanticSqueakAgent»eval: aString
   "Evaluate a Smalltalk expression in the running
Squeak image."
   <function: eval(
      expression: string "e.g. '(8 nthRoot: 3)-1'"
   )>
   ^ Compiler evaluate: aString
```

Finally, the programmer invokes the agent:

```
SemanticSqueakAgent makeNewConversation
   addUserMessage: 'how many windows are open';
   getAssistantReply → 'You currently have 138
open windows in your Squeak environment.'
```

Appendix: SemanticText

Tooling: Conversation Editor

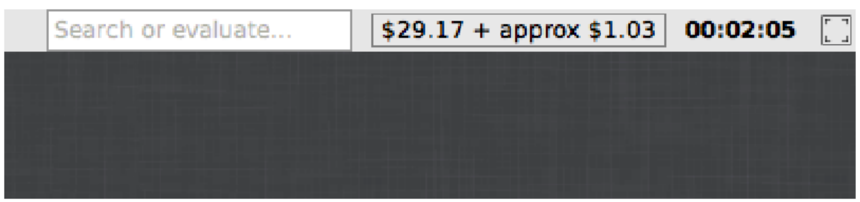


(a) In the *default mode*, end users can engage in conversations with the assistant.

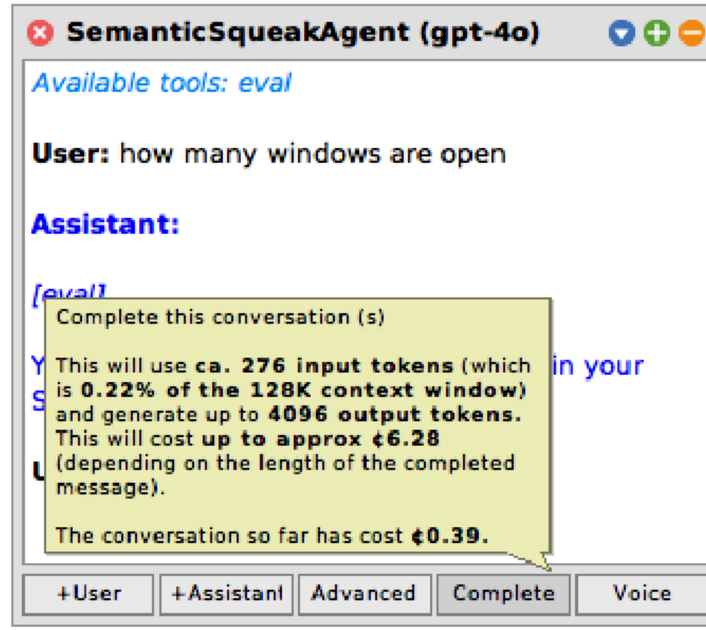
(b) In the *advanced mode*, developers can declare prompts and functions and inspect or simulate function calls of the model.

Appendix: SemanticText

Tooling: Expense Watchers



(a) A global expense watcher attached to the world's main docking bar displays all expenses in the system.



(b) A tooltip in the conversation editor provides detailed information on the prior costs of a conversation and estimates the tokens and price for answering the next user message.

Appendix: Recommendations for Tool Developers

1. **Consider limited accuracy of semantic technologies**
   - Prototype critical model invocations to ensure reasonable and useful responses.
   - Evaluate and communicate risks such as hallucinations.
2. **Design for bidirectional cooperation**
   - Share extensive context and artifacts with language models.
   - Allow programmers to inspect, feedback, and modify agent actions.
3. **Display progress and cost**
   - Reduce experienced latency with progress bars or streamed responses.
   - Offer tools to monitor costs or set rate limits to avoid unexpected expenses.
4. **Optimize semantic applications**
   - Use efficient, smaller, open-source models when possible.
   - Tune prompts, preprocess embedding documents, or fine-tune models.
5. **Collect data early for training and evaluation early on**
   - Log model requests and responses early to support prompt tuning, model evaluation, and fine-tuning.
6. **Address ethical and legal concerns**
   - Understand ethical and environmental concerns and favor responsible and sustainable options.
   - Require users to opt-in, inform them about data usage, anonymize collected data.
   - Best: Require them to build applications from source and bring their own API key.
7. **Consider traditional implementations**
   - Evaluate traditional methods (e.g., parsers, decision trees) and human interventions as alternatives to AI solutions.