Towards Object-centric Time-traveling Debuggers

Maximilian Ignacio Willembrinck Santander[1], Steven Costiou[1], Adrien Vanègue[1] and Anne Etien[2]

[1] *Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL*
[2] *Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL*

Abstract

Object-centric debugging aims at facilitating the debugging of object-oriented programs by focusing debugging operations on specific objects. This technique is tedious to use because developers have to manually find objects to debug, which is not straightforward.

Time-traveling debuggers allows developers to explore executions back and forth in time. It has been shown that time-traveling features effectively facilitate debugging and program understanding.

We propose to combine these techniques to benefit from both of them to debug object-oriented programs. Time-travel navigation could help finding and remembering objects by providing means to explore executions back and forth. Object-centric debugging could extend time-traveling debugging with object-centric exploration features. These techniques have never been combined, and the challenges and benefits of such combination have never been explored.

We present *SeekerOC*, a time-traveling debugger prototype which provides object-centric debugging support. To combine both techniques, we use *Time-Traveling Queries*, a query system to automatically explore executions. We discuss the expected benefits of this combination, and we argue that exploring object-centric time-traveling debugging will open new research perspectives towards more effective debugging techniques and tools for object-oriented systems.

Keywords

Object-Centric Debugging, Time-Travel Debugging, Program Comprehension.

1. Introduction and Motivation: Debugging Objects

To debug their programs, developers use debuggers which traditionally provide a set of standard debugging operations. Some of these debugging operations (*e.g.*, breakpoints) are defined for a class (*e.g.*, in a method) and apply to all instances of that class. Meanwhile, object-centric debugging is a technique proposed to improve the debugging of object-oriented systems by scoping object-centric debugging operations to specific objects [1]. This helps debugging by reducing user interactions to understand bugs [1, 2], and by exposing and hot fixing buggy objects [3]. However in practice, object-centric debugging is difficult to use. Developers have to manually explore their executions to find objects to debug. This is tedious and error-prone. Developers have to repeat many times the same program exploration process. Object-centric debugging operations also generate false positives. For example, an object-centric breakpoint might halt the system many times for a single object. Developers then need to go through numerous haltings

IWST'22: International Workshop on Smalltalk Technologies

until finding useful information for debugging. This is as tedious as manually stepping an execution with a standard debugger.

Time-traveling debugging to the rescue. With time-traveling debuggers, developers are able to deterministically explore a program execution back and forth in time [4, 5, 6, 7]. When they miss a critical point, developers can rewind an execution and replay it from a few steps back. This prevents developers from the hassle of having to restart the entire execution and to repeat the same debugging investigation [8, 9].

We argue that this capability of navigating through time could improve the application of object-centric debugging. Developers would be able to explore the execution to find which object to debug without the risk of missing an interesting moment, and to navigate directly to events of interest for those objects. Furthermore, time-traveling debuggers could provide new support for tracking objects to debug. For example, we could specify what objects we are looking for by using a predicate expression *eg., objects of a specific class, or with specific instance variables values*, and automatically explore the execution back and forth to find those objects. However, to the best of our knowledge, time-traveling debuggers do not support object-centric debugging and vice-versa.

Proposition: combining time-traveling and object-centric debugging. We argue that enhancing object-centric debugging by time-traveling mechanisms would enable new debugging tools (dedicated interfaces, requests, views...) that would improve the debugging of object-oriented programs. In this paper, we illustrate our first probe combining both techniques. We use our execution querying system named *Time-Traveling Queries* [10] (*TTQs*), which automatically and systematically explore an execution to find objects and to define time-traveling debugging operations on them. We present *SeekerOC*, our time-traveling debugger prototype that implements object-centric queries. We discuss the benefits of joining both techniques in terms of debugging capabilities and effect, and why it is worth pushing this research perspective further on.

2. Identifying Particular Objects is Challenging: A Running Example

In the following, we illustrate the difficulties developers go through when they try to comprehend objects behavior during debugging. We explain how object-centric debugging and time-traveling debugging support these challenges, and their limitations. As an example, we use a test method from the Pharo 11 code base:

Listing 1: The split-join test of collections in Pharo 11.

```
SplitJoinTest >> testSplitOrderedCollectionOnOrderedCollection
 self assert: (((1 to: 10) asOrderedCollection) splitOn: ((4 to: 5)
     asOrderedCollection))
     equals: {(1 to: 3) asOrderedCollection. (6 to: 10) asOrderedCollection}
         asOrderedCollection
```

In this test, an `OrderedCollection` of 10 elements is split by another `OrderedCollection` with two elements. This operation is expected to produce a new

`OrderedCollection` containing both left and right sides of the split operation (each side of the split is an `OrderedCollection`). The developer, trying to understand this execution, wants to obtain information about the behavior and state of instances of `OrderedCollection` during the execution of the test. The standard procedure is to step the execution until the desired object is instantiated. This potentially requires numerous and carefully executed steps before the observation of the desired object can take place. Moreover, the execution of this test produces several instances of `OrderedCollection`, and these instantiation calls are not immediately visible in the test code. It is difficult to track each object, as none of them is seemly stored in a variable. This makes such manual approach even more tedious.

Using an object-centric debugger, developers can improve over such approach by using specific breakpoints that halts the execution whenever a class is instantiated. With this technique, developers need to halt a certain number of times to reach the `OrderedCollection` of interest. Once the object of interest has been reached, developers can use object-centric breakpoints to observe the object's behavior and the evolution of its state. One downside of this technique is that it might take many halts to reach the object of interest in the first place, and then several other object-centric breakpoints to observe a relevant information. If the developer accidentally misses that relevant information, then all this procedure has to be restarted from scratch.

By using a time-traveling debugger, we can enhance this approach by navigating back and forth in the execution. If developers miss the observation of one of the collections, or if they passed it and want to observe it again, they reverse the execution and look for that collection, without the need to redoing everything from the beginning.

Object-centric debugging and time-traveling debugging both provide enhancements over standard debugging tools on their own. We argue that they can complement each other to improve debugging further. In the following, we start to explore how these techniques can benefit from each other, what is required to enable this new joint approach and how could we materialize this approach into a tool.

3. Debugging Objects Through Time with *SeekerOC*

In this section we present our debugger *SeekerOC*. We explain how it helps solving the difficulties identified by using debugging queries and new specialized tools.

3.1. *SeekerOC*: Debugging with Object-centric *Time-Traveling Queries*

Our debugger offers a list of general purpose queries that can be conveniently executed from a context menu. We extended this menu with new queries to ease the task of identifying and tracking objects, as described next.

Figure 1 shows our debugger opened at the beginning of the execution of the code from Listing 1. To find all collections created during the execution of that code, we select the `OrderedCollection` string and we execute the query *All instances creation of class named as selection* from the context menu (Figure 1(a)). Once the query has been executed, the results are displayed in a table (Figure 1`(b)`). These results contain

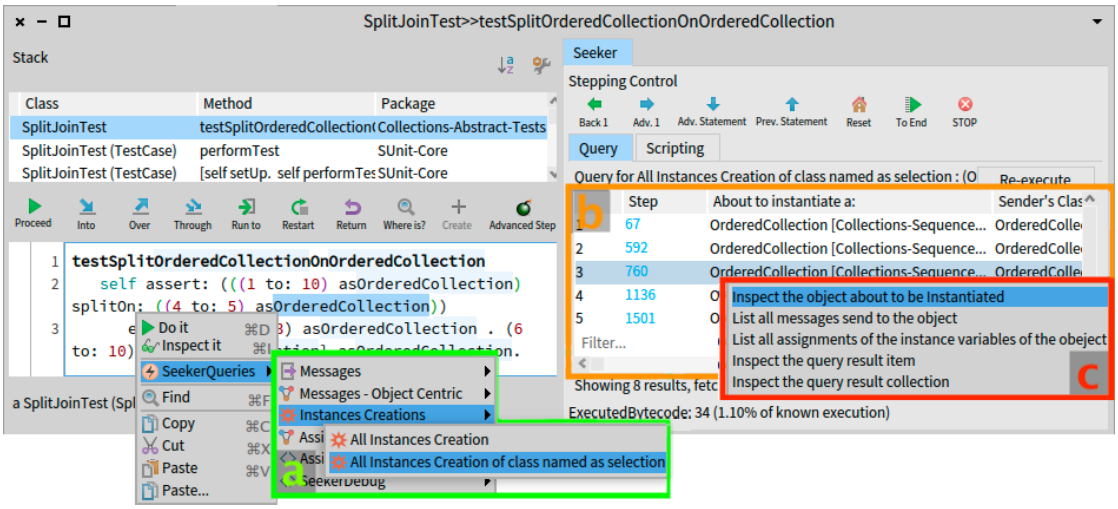

Figure 1: *SeekerOC* interface. To the left, the *Time-Traveling Queries* menu(a) available in the code presenter. To the right, the Query Results Table(b) and its context menu(c).

the complete list of all the `OrderedCollection` objects that are instantiated during the execution of the debugged program.

From the results table, we effortlessly have access to each one of these objects by right clicking the corresponding result item, and choosing the command *inspect object about to be instantiated* (Figure 1`(c)`). This command opens an inspector on the object corresponding to the selected result row, and shows the object state in the execution state from which we executed the query. If the object has not yet been instantiated, the execution is advanced to such event, so the object can be inspected.

Every result entry of *TTQs* includes a timestamp (the step number) of every registered event. Clicking in the `step` column of any result item will advance or reverse the execution, time-traveling to the registered timestamp. This feature makes queries results act as *bookmarks* of an execution, providing a practical and precise alternative to using breakpoints and tedious stepping operations for program exploration.

3.2. The *SeekerOC Inspector*: A Promising Object-centric Querying Hub

Pharo traditionally offers a graphical utility for object inspection — a view that displays information about the state of an object, its API, and other useful information. During a *SeekerOC* debugging session, inspecting any object of the debugged execution will open an enhanced version of the Pharo *Inspector*.

This specialized inspector synchronizes with the time-traveling debugger to keep an updated view of the inspected object. Stepping the execution forward or backwards triggers updates of the displayed information. This inspector contains a menu for object-centric queries available for the inspected object (Figure 2):

- *Time travel to instantiation instruction.*
- *List all messages sent to the inspected object.*

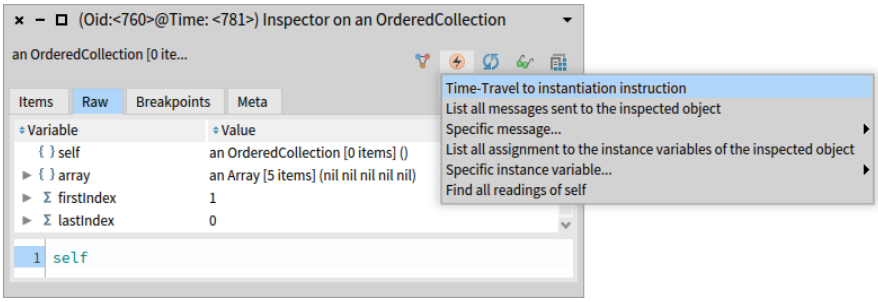

Figure 2: Object-centric *time-traveling queries* menu in the *SeekerOC* inspector: a practical access to relevant object-centric *TTQs*.

- Submenu *Specific message...*: displays every message selector that the object responds to. Clicking any particular listed selector launches a query of the type: *List all messages sent with that specific selector to the inspected object.*
- *List all assignments to the instance variables of the inspected object.*
- Submenu *Specific instance variable...*: displays every instance variable of the object. Clicking any particular listed variable name launches a query of the type: *List all assignments to that specific instance variables of the inspected object.*
- *Find all readings of self.*

Every *TTQ* provides a set of timestamp-indexed results in the *Query Results Table*. Therefore, by clicking on the *step* column of every result item, we time-travel to the timestamp of each registered event, without the inconveniences of stepping or halting.

By using *TTQs*, the relevant debugging information of the execution is collected effortlessly. This information is now directly accessible from the *Query Results Table*, instead of requiring developers to go through the traditional manual debugging process.

4. Implementation

We built our experimental object-centric prototype *SeekerOC* upon *SeekerDebugger*, our queryable time-traveling debugger implemented for Pharo. *SeekerDebugger* provides the base mechanism to perform general purpose *Time-Traveling Queries*[10] over an execution. *SeekerOC* improves upon it, offering new specialized queries that focus on objects and their events. *SeekerOC* and its implementation details are available online[1].

Determinism: A hard requirement. A requirement for object-centric queries to work is that the program must be re-executed deterministically. Determinism is ensured by our debugger backend. Instantiated objects' identities will remain constant upon multiple executions. We can therefore write queries about those objects without being concerned that objects have a different identity after being reinstantiated during a re-execution.

[1] https://github.com/Willembrinck/SeekerOC-2022/

5. Discussion: Towards Object-centric Time-traveling Debuggers

Our work materializes an exploration step towards a new type of debugging tool: an object-centric time-traveling debugger, based on debugging queries. In our proposition, *Time-Traveling Queries* play the central role joining object-centric and time-traveling debugging. Querying an execution offers a traced representation of it, granting easy access to objects. The time-traveling back-end ensures that the execution can be re-executed following the same trace. Therefore, if an object can be identified in such trace, it can be referenced in any re-execution of the program. This generates opportunities to explore object-centric debugging in an new joint debugging paradigm, enabling techniques and features previously unfeasible.

These opportunities open many questions. Are *TTQs* the only mechanism required to connect both techniques? If not, what are the required properties and/or mechanisms to build object-centric time-traveling debuggers? What precisely are the benefits of such debugger, and to what extent do they benefit to the debugging activity? In future work, we plan to investigate these questions by pursuing our implementation of object-centric *TTQs* and analyzing their requirements and effectiveness through empirical experiments.

6. Related Work

Object-centric debugging and time traveling debugging have been the subject of research in the recent years. However, and to the best of our knowledge, they have never been combined to support each other. Here we discuss how some of these works relate to ours.

Object-centric debuggers. Object Miners [11] is a set of object-centric tools to acquire, capture and replay objects from specific expressions of a program. Developers select (sub)expressions from the program from which objects are automatically captured and debugged during the execution. Specific objects' state can be recorded, then replayed and traversed to observe the evolution of that state. While the tool allows for exploring the past state of an object, it does not support time-traveling operations.

Other work [12, 13] keep track of changes in objects during the execution of programs. Developers can visualize past states of objects and their behavior. This is a post-mortem approach without time-traveling capabilities.

Time-traveling debuggers. *Expositor* [4] combines scripting and time-travel debugging to allow programmers to automate complex debugging tasks. From an execution, *Expositor* generates traces that developers manipulate as lists with operations such as map and filter. Our query model provides a similar behavior, where execution traces can be created, operated, and evaluated to generate results. However, *Expositor* does not provide a tool to visualize and exploit results, and does not support object-centric operations.

Whyline [14, 15] offers contextual queries for program comprehension. It offers certains object-centric queries, but they are difficult to extend. In contrast, *SeekerOC* is extensible through user defined *TTQs*, giving developers the means to create new specialized queries.

7. Conclusion and Perspectives

In this paper we argued that time-traveling debuggers could provide support to the practical application of object-centric debugging. We explored this concept by building *SeekerOC*, a time-traveling debugger using *Time-Traveling Queries*. *TTQs* provide the means for quick execution exploration which enhances developers capabilities for seeking the important object-related events during debugging. Our queries include information in their results that facilitates object identification, enabling our new proposed specialized object-centric tools, such as the improved object *Inspector* and its object-centric *TTQs* menu. Due to the importance of *TTQs* in our proposition, an interesting prospect of research is to explore what new queries can boost object-centric debugging activities.

Acknowledgments

This work is funded by the ANR project OCRE https://anr.fr/Projet-ANR-21-CE25-0004 and Inria.

References

[1] J. Ressia, A. Bergel, O. Nierstrasz, Object-centric debugging, in: Proceeding of the 34rd international conference on Software engineering, ICSE '12, 2012.

[2] C. Corrodi, Towards efficient object-centric debugging with declarative breakpoints, in: SATToSE 2016, 2016.

[3] S. Costiou, Unanticipated behavior adaptation : application to the debugging of running programs, Ph.D. thesis, Université de Bretagne occidentale - Brest, 2018.

[4] K. Y. Phang, J. S. Foster, M. Hicks, Expositor: Scriptable time-travel debugging with first-class traces, in: 2013 35th International Conference on Software Engineering (ICSE), 2013, pp. 352–361.

[5] Y. Wang, H. Patil, C. Pereira, G. Lueck, R. Gupta, I. Neamtiu, Drdebug: Deterministic replay based cyclic debugging with dynamic slicing, in: Proceedings of annual IEEE/ACM international symposium on code generation and optimization, ACM, 2014, p. 98.

[6] P. Montesinos, L. Ceze, J. Torrellas, Delorean: Recording and deterministically replaying shared-memory multiprocessor execution efficiently, ACM SIGARCH Computer Architecture News 36 (2008) 289–300.

[7] C. Zamfir, G. Altekar, G. Candea, Debug determinism: The sweet spot for {Replay-Based} debugging, in: 13th Workshop on Hot Topics in Operating Systems (HotOS XIII), 2011.

[8] E. T. Barr, M. Marron, Tardis: Affordable time-travel debugging in managed runtimes, in: Proceedings of International Conference on OOPSLA'14, volume 49, ACM, 2014, pp. 67–82.

[9] J. Hoey, I. Lanese, N. Nishida, I. Ulidowski, G. Vidal, A case study for reversible computing: Reversible debugging of concurrent programs., 2020.

[10] M. Willembrinck, S. Costiou, A. Etien, S. Ducasse, Time-traveling debugging queries: Faster program exploration, in: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS), 2021, pp. 642–653.

[11] S. Costiou, M. Kerboeuf, C. Toullec, A. Plantec, S. Ducasse, Object miners: Acquire, capture and replay objects to track elusive bugs, Journal of Object Technology 19 (2020) 1:1–32.

[12] A. Lienhard, S. Ducasse, T. Gîrba, O. Nierstrasz, Capturing how objects flow at runtime, in: Proceedings International Workshop on PCODA'06, 2006, pp. 39–43.

[13] A. Lienhard, T. Gîrba, O. Nierstrasz, Practical object-oriented back-in-time debugging, in: Proceedings of the 22nd ECOOP'08, volume 5142 of *LNCS*, Springer, 2008, pp. 592–615.

[14] A. J. Ko, B. A. Myers, Designing the whyline: a debugging interface for asking questions about program behavior, in: Proceedings of the 2004 conference on Human factors in computing systems, ACM Press, 2004, pp. 151–158.

[15] A. J. Ko, B. A. Myers, Debugging reinvented: Asking and answering why and why not questions about program behavior, in: In Proceedings of the 30th ICSE 08, 2008.