Breaking execution when two particular objects interact with each other allows significant gain of user interactions for finding the source of bugs in specific scenarios. Breaking execution when a particular object receives a message provides a way to suspicious objects and activate only when those particular objects are in the right context. The strategy of the Centre is to develop an internationally renowned centre of excellence with a significant impact on the City of Lille and its surrounding area. It works to achieve this by pursuing a range of ambitious research projects in such fields of computer science as the intelligence of data and adaptive software systems. Building on the synergies between research and industry, Inria is a major contributor to skills and technology transfer in the field of computer science.

Context and employment

The goal of this PhD is to study Object-Centric Debugging, a young but promising technique for the debugging of object-oriented programs. The student will join the debugging group within the RMoD team, in the context of the ANR 2021 project OCRE: Object-Centric debugging Reloaded.

RMoD

The goal of RMoD is to support remodularization and development of modular object-oriented applications. We tackle this objective from two complementary perspectives: reengineering, and constructs for dynamic languages. In the context of languages, we revisit language concepts such as modules and composition; we are also working on a new generation of reflective systems. We experiment with these programming constructs using Pharo, an open source, reflective, object-oriented language. http://rmod.inria.fr

Mission statement

When debugging object-oriented programs, we often identify a suspicious object that needs to be investigated. However, mainstream debuggers only provide a call-stack based perspective that shows the executed code in sequential order (which code called which code). From stack based perspectives, it is extremely difficult to know where to apply debugging operations because we cannot know at run time where suspicious objects are used in the code.

The problem is that, at run time, objects are passed around and interact with each other through standard method activations. A buggy object might touch the state of another object (the affected object) that later produces the visible effects of the bug. When that effect becomes visible, traditional debuggers only show the affected execution stack, where the buggy object and its past interaction with the affected object are not observable anymore.

This distance between the affected object (in the active stack) and the buggy object (in a past, terminated stack) hides the interaction between the two objects and hinders debugging. Stack based perspectives do not provide enough information to decide where and how to apply debugging operations in the program.

This problem becomes harder in complex programs. Hundreds of instances from the same class may execute behavior within isolated execution stacks, but only one of them may cause the bug. It is already difficult to understand that the effect of a bug observed from the current execution stack comes from an object in another, past execution stack. Now, we also have to differentiate this particular object from lots of similar objects in order to debug it. Traditional tools such as breakpoints are impractical to use: it is impossible to break the execution for each one of those objects. To debug only the right object, developers have to insert complex conditional instructions into the source code to manually filter the objects. In the case of hard bugs, it is common that they cannot express those conditions due to a lack of information or tools. These difficulties also render other approaches, such as back-in-time debuggers, ineffective.

This is where object-centric debugging comes into play, with the capability of breaking execution for a single specific object [1, 2]. Object-centric debugging formulates the hypothesis that, by focusing the scope of debugging (views, interactions, operations) on singular objects, tracking hard bugs and finding their source would be easier. Object-centric debugging operations are applied only to suspicious objects and activate only when those particular objects are in the right context. Breaking execution when a particular object receives a message provides a significant gain of user interactions for finding the source of bugs in specific scenarios [1, 2]. Breaking execution when two particular objects interact with each other allows...
for a program comprehension that would be hard to acquire with standard breakpoints [1]. Modifying single objects’ behavior in running programs helps in exposing bugs and finding their source, and allows developers to hot fix buggy objects [4].

Object-centric debugging has the potential to drastically lower the cost of tracking and understanding hard bugs, and thus to become a strong reference technique for the debugging of object-oriented programs. However, the technique is still in its infancy and its general hypothesis has not been completely tested because it requires to first solve fundamental and practical problems:

(1) **How to find objects to debug** among thousands or millions, especially when faulty objects are not available from the execution stack where the effects of bugs become visible?

(2) Object-centric debugging lacks practical and empirical feedback and evaluation, and there is, as of today, no large-scale empirical evaluation testing the promises of the technique.

The objective of this project is to achieve and to understand the full potential of object-centric debugging by addressing these problems. Our findings will set the first generation of object-centric debuggers, and make them available for every object-oriented developer through:

- Formal and conceptual descriptions including the new solutions for (1);
- Example implementations to prove the feasibility of the solutions and to evaluate them (2);
- Systematic debugging methods to choose when and how to use object-centric debuggers (from (2)).

References


Principales activités

The plan is to:

- Join the team work around debugging, that includes 3 researchers and 1 phd student from the RMoD team at Inria Lille, a collaboration with Thales DMS Brest, the SMartse team from University of Chicoutimi At Quebec and with the University of Zurich.
- Survey the key developer activities around debugging
- Define new models and solutions
- Realize and experiment prototypes, using Pharo ([www.pharo.org](http://www.pharo.org))
- Design and run large-scale empirical evaluations with real end-users and practitioners
- Publish results in top venues

Compétences

Technical skills and level required

- OOP, TDD
- Reflective programming
- Program transformation

Language

- English

Relational skills

- Good team work skills

Other valued appreciated

- Knowledge of the Pharo language is a plus
- A strong interest about debugging and dynamic languages
- A strong will to learn and to explore new things

Avantages

You will integrate a dynamic team composed of international scientific experts in the domain of software engineering ([https://rmod.inria.fr](https://rmod.inria.fr)).

You will work on bleeding-edge research topics with international academic and industrial partners recognized in the field of reverse-engineering, software evolution, virtual machines, dynamic languages, reflection and debugging.

You will have the opportunity to integrate the Pharo community, and participate to the Pharo sprints held every month in the RMoD's offices.

This will give you great opportunities to link to the community, an learn from world-class object-oriented languages experts.

You can also benefit from free technical and more general on-site training.

For international candidates, our administrative services helps you in many different administrative tasks (visa, residence permit, social security, accommodations, bank...).
You will work in a stimulating and engaging work environment:
- on-site catering with discounts
- reimbursement of transportation (50%)
- remote working (90 days per year)
- paid leave: 9 weeks a year, with additional facilities for exceptional cases (e.g., kids)
- social, cultural and sport services and discounts through the local Inria association

Rémunération
1st and 2nd year : 1 982€ gross monthly salary (before taxes)
3rd year : 2 085€ gross monthly salary (before taxes)