The thesis will consider code migration at different levels of abstraction and complexity. The plan is to treat each level of abstraction independently from the other in successive phases of the thesis so that one abstract issue will not pollute another simpler one (“Migrate then Redesign” approach to treat each level of abstraction separately). The thesis will consider code migration at different levels of abstraction and complexity. The plan is to treat each level of abstraction independently from the other in successive phases of the thesis so that one abstract issue will not pollute another simpler one (“Migrate then Redesign” approach to treat each level of abstraction separately).
• **Idioms:** All programming languages have typical ways of doing things that must be respected for the code to look “natural” ([Fel90, BRPP10]) (e.g. iterating through a table by using a pointer in C). For the final code to be easily understandable and maintainable, it should follow these standards. This will be achieved by having transformation rules applied on the already converted C code. To follow up on the simple example of manipulating arrays in C, we will identify patterns of FORTRAN code (a loop manipulating a table) that should be transformed and provide a flexible modification of the Abstract Syntax Tree (AST) to transform it in a more natural C/C++ code.

• **Libraries:** Any application depends on some library that needs to be replaced in the migrated code. Such replacement can be one-to-one in the best case, but may often require to develop wrappers, or a new component (when no satisfactory substitution can be found). In our case, we will be helped by the fact that Fortran and C can easily interact. The FORTRAN code already uses some C libraries and the C/C++ code will initially use some FORTRAN libraries. Thus we plan to keep the original libraries in a first time, and only after migration, replace the Fortran libraries by C/C++ ones as the programmers will see fit.

• **Patterns of optimization:** We will look for domain-specific patterns of code that could allow for advanced optimizations of the C code or opportunities for parallelization. Again, this will rely on AST pattern matching with modification of the AST to get optimized code. The advantage of this solution is that it can be continued to be used after the end of the project as new optimization patterns are discovered. RMod has an ongoing thesis (Jason Lecerf, in collaboration with CEA) on this subject (e.g. [RDRL17]).

• **Pardigm:** Transforming procedural programs into object-oriented ones is an old challenge of software reverse engineering (e.g. [NK95, CCLL01]). Past research focused a lot on full automation and source code analysis. This approach has its limits as the code representing possible classes is often intertwined with other unrelated code. We will offer an interactive environment, where experts of the application can propose domain concepts that are expected to be encountered and look for beacons of these concepts in the source code (e.g. in identifiers or in comments). From the initial set of code entities thus recovered, the experts will be able to add other entities that are highly related to the initial ones through several query mechanisms. The process will be iterative with a gradual refinement of the understanding of the concepts and their implementation. Once the expert is satisfied with the identified set of functions and data, the tool will modify the source code to replace these C entities by C++ classes.

Note that Grosse-Kunstleve et al. [GKTS12] made an opposite choice and tried to “eliminate the need for manual work”. But this was at the cost of the quality of the OO code, where classes are generated from Fortran programs which are a too coarse level of abstraction.

On the other hand, we successfully used an iterative and manual approach, with gradual refinement, in the Synectique company, for one of its clients, to help split huge classes (thousands of lines of code) into smaller ones.

**Bibliography**


**Main activities**

The PhD student will need to:

• spend some time with the OMN developers group to gain insider understanding of the application domain constraints and get acquainted with the Fortran programming language and the Code_Carmel application itself;

• conduct a literature survey on programming language migration, procedural to OO migration, and Fortran to C/C++ migration;

• propose solutions to the Code_Carmel migration problem and implement them as tools...
within the Moose software analysis platform; write scientific article to disseminate the results of the research.

Skills

- Programming and particularly Object-Oriented programming
- Some mathematical background would help in understanding the constraints of the application domain
- Good English writing skills
- A preliminary knowledge of the Pharo programming language is not required but would be an advantage.

Benefits package

- Subsidised catering service
- Partially-reimbursed public transport
- Social security
- Paid leave
- Sports facilities
- Flexible working hours

More information about Lille:
http://www.lille3000.eu/portail/
http://www.lillemetropole.fr/mel.html

Remuneration

- The gross monthly salary is 1982€ for the 1st and 2nd year, 2085€ for the 3rd year.