Pragmatics of Rust and C++: The implementation of a window manager

Max van Deurzen
Technische Universität München
Munich, Germany
m.deurzen@tum.de

***Abstract*—In comparing and discussing programming languages (and natural languages, for that matter), not only *syntax* and *semantics* are of importance. *Pragmatics* is the third general area of language description, which deals with the practical aspects of how language constructs and features allow its users to achieve various objectives. In this paper, we will look at the pragmatics of both Rust and C++. Specifically, we will be comparing the languages in their ability to be used as a tool to write medium to large *system programs*. As a case study, we will be discussing two implementations of a complete ICCCM and EWMH compliant top-level reparenting, tiling window manager, built on top of the X Window System: one written in Rust, and the other in C++.**

***Keywords*—Pragmatics, Rust, C++, Window Manager**

I. Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

II. External Dependency Management

As a programming language's ability to aid the programmer in managing external dependencies—by, for instance, providing various tools that come installed with its compiler or development environment—is generally not incorporated into that language's syntax or, by extension, its semantics, it is traditionally not considered an aspect of that language's feature set per se. Notwithstanding, it is more and more becoming an appreciated addition to the *ecosystem* around a language, especially so for compiled languages. In fact, many would consider automated external dependency management to be a must for any modern programming language. As it directly affects both the portability of code, as well as the (ease of) managing different versions of a dependency, a language's ability to unburden the programmer from the manual management of external dependencies can greatly improve the maintainability of a project, and can therefore indeed be viewed as a feature of the language in light of its *pragmatics*. In this section, we will be discussing the practicalities of working with external code in both Rust and C++. We will do this by means of a comparison between two window manager implementations, one written in Rust, and the other written in C++, which we will henceforth be referring to as WMRS and WMCPP, respectively.

Both window managers are built on top of the X Window System, which means they rely on an external library to interface with the X server. Although WMRS and WMCPP each use a different library to achieve this (XCB over `libxcb` and Xlib over `libX11`, respectively), the concept is the same, as they both require the importing of an external body of code.

A. Portability

C++ does not come with an external dependency manager. That is to say, the official ISO standard C++ specification [1] does not outline the functionality or otherwise existence of any package manager. As a result, developers have no other choice than to resort to third-party tools to manage dependencies, to reinvent the wheel themselves and hack together configure and build scripts that take external dependencies into account, or to disregard package management altogether and let other developers and end users resolve dependencies on their own. In any case, C++ dependency management is complex, and the lack of a standardized tool inhibits the adoption of C++ projects, as well as their portability to other platforms and development environments.

Since the only external dependency WMCPP is reliant on is `libX11`, which is fairly ubiquitous and readily available through distributions' own package managers, it merely links with the X11 development libraries without the use of any form of own dependency management. Specifically, building the project is done using the following **`Make`** script (large parts altered or redacted for clarity).

```
CC = g++
CXXFLAGS = -std=c++17 -march=native -O3
LDFLAGS = `pkg-config --libs x11` -flto

obj/%.o: src/%.cc
  ${CC} ${CXXFLAGS} -MMD -c $< -o $@
release: obj/%.o
  ${CC} $< ${LDFLAGS} -o bin/wmCPP
```

Assuring the availability of an external body of code is not the only aspect of external dependecy management that affects the pragmatics of C++. Writing a library that is to be imported as an external dependency and including code from an external dependency both require special care to be taken to make sure no symbol collisions occur. In header files, this is usually done by making use of so-called *header guards*, a set of preprocessor directives that render including a header an idempotent operation, preventing double inclusion errors. If incorrect double inclusion does unexpectedly occur, for instance due to collisions in the header guards themselves, vague—as header guards are handled by the preprocessor, not the compiler—and hard-to-trace compiler errors arise, subjecting the programmer to unnecessary mental strain, and wasting time. Although not part of the official standard, many C++ compilers support the `#pragma once` preprocessor directive, providing the same functionality as header guards, but preventing the occurrence of double inclusion errors.

Another source of issues is the order of include directives in C++. The ordering of included files relative to one another can significantly influence the outcome of the compilation step, possibly causing errors, one of which being the introduction of circular dependencies while there actually aren't any there (e.g. a forward declaration that was included only after the circular dependency was already introduced).

The Rust programming language has its own package manager, called Cargo. Cargo can automatically download a project's external dependencies (and, transatively, *their* dependencies), compile them, and install them locally, such that they can be used during the linking stage of the build process. To make proper use of Cargo within a Rust project, that project must be turned into a so-called *package*. Rust packages are simply a collection of source files along with a manifest file (named `Cargo.toml`) in the project's root directory. This manifest file describes the package's meta-information (such as its name and version), and a set of *target crates*. A crate is the source code or compiled artifact of either a library or executable program, or possibly a compressed package that is grabbed from a registry (a service that provides a collection of downloadable crates). A package's manifest file describes each of its target crates by specifying their type (binary executable or library), their metadata, and how to build them.

The following manifest file describes the package that represents our Rust window manager, WMRS (parts redacted for clarity).

```toml
[package]
name = "wmRS"
version = "0.1.0"
authors = ["deurzen <m.deurzen@tum.de>"]
edition = "2018"
license = "BSD3"
documentation = "https://docs.rs/wmRS"
readme = "README.md"
default-run = "wmRS"
description = """
An ICCCM & EWMH compliant X11 reparenting,
tiling window manager, written in Rust
"""

[profile.release]
lto = true

[lib]
name = "winsys"
path = "src/winsys/mod.rs"

[[bin]]
name = "wmRS"
path = "src/core/main.rs"

[[bin]]
name = "wmRSbar"
path = "src/bar/main.rs"
required-features = ["bar"]

[[bin]]
name = "wzrdclient"
path = "src/client/main.rs"
required-features = ["client"]

[features]
bar = []
client = []

[dependencies]
x11rb = {
  version = "0.8.0",
  features = [
    "cursor",
    "xinerama",
    "randr",
    "res"
  ]
}
strum = {
  version = "0.19",
  features = ["derive"]
}
strum_macros = "0.19"
anyhow = "1.0.33"
log = "0.4"
simplelog = "0.8.0"
nix = "0.19.0"
```

Our package consists of a single library, along with three binary executables. The library is an abstraction above and wrapper around the interface with the underlying windowing system. It defines a single Rust trait that represents the connection between the window manager and the windowing system. This decouples the implementation of the window manager from that of the interface with the windowing system, and additionally allows for the seamless transition from one windowing system to another, as multiple windowing systems can be targeted by implementing the trait, effectively creating a new wrapper around an external library that directly interfaces that windowing system. Currently, only the interface with the X Window System is implemented, but interfacing with the newer and more modern Wayland is as easy as implementing a new connection to it. A small portion of WMRS's winsys library connection trait looks as follows.

```
pub trait Connection {
  fn flush(&self) -> bool;
  fn step(&self) -> Option<Event>;
  fn connected_outputs(&self) -> Vec<Screen>;
  fn top_level_windows(&self) -> Vec<Window>;
  fn get_pointer_position(&self) -> Pos;
  fn warp_pointer(
    &self,
    pos: Pos,
  );

  fn init_window(
    &self,
    window: Window,
    focus_follows_mouse: bool,
  );
  fn init_frame(
    &self,
    window: Window,
    focus_follows_mouse: bool,
  );
  fn init_unmanaged(
    &self,
    window: Window,
  );
  fn cleanup_window(
    &self,
    window: Window,
  );
  fn map_window(
    &self,
    window: Window,
  );
  fn unmap_window(
    &self,
    window: Window,
  );
  fn reparent_window(
    &self,
    window: Window,
    parent: Window,
    pos: Pos,
  );
  fn unparent_window(
    &self,
    window: Window,
    pos: Pos,
  );
  fn destroy_window(
    &self,
    window: Window,
  );
  fn close_window(
    &self,
    window: Window,
  ) -> bool;
  fn kill_window(
    &self,
    window: Window,
  ) -> bool;
  fn place_window(
    &self,
    window: Window,
    region: &Region,
  );
  fn move_window(
    &self,
    window: Window,
    pos: Pos,
  );
  fn resize_window(
    &self,
    window: Window,
    dim: Dim,
  );
  fn focus_window(
    &self,
    window: Window,
  );
  fn stack_window_above(
    &self,
    window: Window,
    sibling: Option<Window>,
  );

  // ...
}
```

The three binary executable crates represent the window manager, a client program to communicate with the window manager (to control various window management activities, such as closing the currently focused window, from scripts or the command line), and a status bar that displays information about the state of the window manager, such as the currently activated workspace.

B. Versioning

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

III. Build System

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

IV. Main Event Loop

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

A. Internal Events

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

B. Event Dispatch

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

V. Communication with the Environment

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

A. Inter-Process Communication

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

VI. Key Bindings

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit

blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

VII. Clients

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

A. Reference Management

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

B. State

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

VIII. Workspaces

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

IX. Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

References

[1] ISO/IEC. (2020). “ISO International Standard ISO/IEC 14882:2020(E) – Programming Language C++.” [Working draft]. Geneva, Switzerland: International Organization for Standardization (ISO). Retrieved from https://isocpp.org/std/the-standard