Contents

Abstract **iii**

1 Introduction **1**

1.1 Context and Motivation . . . 2

1.1.1 Configuring complex systems (esempi docker, k8s, alchemist) 2

1.1.2 Challenges and limitations (esempi di errori/difficoltà) . . . 2

1.1.3 The Need for a Type-Safe Configuration Language . . . 2

1.2 Background . . . 2

1.2.1 Alchemist . . . 2

1.2.2 Domain-Specific Languages . . . 8

1.2.3 Kotlin DSLs . . . 8

1.2.4 Kotlin Symbol Processing . . . 8

2 Analysis **9**

2.1 High-level Goals . . . 9

2.2 Constraints . . . 10

2.2.1 Backward Compatibility . . . 11

2.2.2 Kotlin Language Constraints . . . 11

2.3 Functional Requirements . . . 12

2.3.1 YAML Equivalence . . . 12

2.3.2 Loading System Requirements . . . 12

2.3.3 Type Safety . . . 15

2.4 Non-functional Requirements . . . 16

2.4.1 DSL API Usability . . . 17

2.4.2 Extensibility . . . 17

2.4.3 Performance . . . 17

2.5 Requirements Analysis . . . 17

2.6 Problem Analysis . . . 17

2.6.1 Limitations of YAML Configuration . . . 17

2.6.2 Legacy Type System Limitations . . . 17

2.7 Simulation Configuration Domain . . . 17

3 Design **19**
3.1 Architecture 19
3.1.1 Scoped Information Contexts 19
3.1.2 Loading System 19
3.2 Detailed Design 19
3.2.1 Task-Based System 19
3.2.2 DSL Builder Structure 19

4 Implementation **21**
4.1 DSL Implementation 21
4.1.1 Builder Functions 21
4.1.2 Variables and Constants 21
4.1.3 Batch Mode Support 21
4.2 KSP Implementation 21
4.2.1 Symbol Processing Pipeline 21
4.2.2 Code Generation Strategy 21
4.2.3 Context Injection Mechanism 21

5 Evaluation **23**
5.1 Testing 23
5.1.1 Unit Testing 23
5.1.2 Integration Testing 23
5.1.3 DSL Equivalence Testing 23
5.1.4 Performance Testing 23
5.2 Experimental Evaluation 23
5.2.1 DSL Usability 23
5.2.2 KSP Processor Benefits 23

6 Conclusion and Future Work **25**

27

Bibliography **27**