**Technische Universität München**
Institut für Informatik
Lehrstuhl für Rechnertechnik und Rechnerorganisation

# Automatic Performance Engineering Workflows for High Performance Computing

*Ventsislav Petkov*

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

| | | |
|---|---|---|
| Vorsitzende(r): | | Univ.-Prof. Dr. Helmut Krcmar |
| Prüfer der Dissertation: | | |
| | 1. | Univ.-Prof. Dr. Hans Michael Gerndt |
| | 2. | Univ.-Prof. Dr. Felix Gerd Eugen Wolf |
| | | Rheinisch-Westfälische Technische Hochschule Aachen |

Die Dissertation wurde am 25.09.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 03.02.2014 angenommen.

# Acknowledgments

The long journey in successfully completing this dissertation has been an unforgettable experience for me. Looking back, there are a number of people without whom this work might not have been written and whose tremendous support I greatly appreciate.

I would like to express my greatest gratitude to Professor Michael Gerndt for his insightful guidance, valuable support and unmitigated encouragement through my doctoral studies. I am thankful to him for giving me the complete freedom to follow my research and constantly providing me with insightful comments and suggestions. Furthermore, I would like to thank him for giving me the great chance to work in such a supportive environment like the Chair of Computer Architecture (LRR) at Technische Universität München (TUM).

I would also like to offer my special thanks to my second supervisor, Professor Felix Wolf, for his time reviewing this dissertation. I am also very grateful to him for giving me the opportunity to be part of the leading Virtual Institute - High Productivity Supercomputing (VI-HPS) and to work within the Performance Dynamics of Massively Parallel Codes (LMAC) project.

Furthermore, I like to express my gratitude to all my friends and colleagues at LRR-TUM for their valuable support and all the great moments that we spent together throughout the last years.

Last but not least, I would like to express the deepest appreciation to my mother Ani, my father Valeriy and my brother Martin for their continuous, unwavering support over so many years and for constantly being the driving force to materialize this dissertation. I would also like to thank my girlfriend Martina for her loving care and affection which immensely helped me to stay focused on achieving my goal.

*Ventsislav Petkov*
*Munich, Germany*
*September 2013*

# Abstract

During the typical performance engineering process, application developers must often complete multiple iterative tasks to get an in-depth runtime profile of their application in order to identify new optimization opportunities and implement an effective tuning strategy. However, the majority of today's mainstream tools do not support common workflows like analyzing the scalability and stability of parallel applications, studying their dynamic behavior, optimizing them for a particular architecture or even improving their power efficiency to help lower the operational costs of High-Performance Computing (HPC) centers. This, combined with the high complexity of today's computing systems, makes it extremely difficult for the developers to collect, maintain, and organize data from numerous performance engineering experiments and simultaneously track the evolution of their code throughout the overall software development and tuning cycle.

However, problems such as long-running iterative processes, diverse data sources and totally different environments exist also in other research areas. Making relevant data-based decisions that lead to improving process efficiency has been the goal of business intelligence (BI) for many years now. A lot of research has been done in this field in order to accelerate and improve the overall decision-making process and create a smooth and stable working environment. Moreover, many tools have been developed to support the processing of huge data quantities and create model-driven, easier to use systems that greatly automate the overall analysis process.

Consequently, this dissertation explores the concept of process automation in the field of parallel performance engineering and proposes a framework to support application developers in structuring and executing the overall tuning process and simultaneously tracking the evolution of the source code that is part of it. It adopts established standards and research ideas from the field of business intelligence and adapts them to the scientific domain of performance analysis and tuning of HPC applications.

Furthermore, this work discusses new extensions to the Periscope performance engineering tool to accommodate such automated workflows and create a flexible, cross-platform environment. As a result, the overall performance analysis and tuning process can be enhanced and thus the productivity of high-performance computing developers can be improved. The evaluation of the thesis uses a set of popular performance engineering workflows, HPC benchmark codes and real-world applications to show the feasibility of the proposed framework.

# Zusammenfassung

Während des typischen Performance Engineering Prozesses, müssen Anwendungsentwickler oft mehrere iterative Aufgaben durchführen um ein vollständiges und detailliertes Laufzeitprofil ihrer Anwendung zu erhalten. Dieses Profil kann dann eingesetzt werden um neue Verbesserungsmöglichkeiten zu identifizieren und eine optimale Optimierungsstrategie zu entwickeln. Allerdings hat die Mehrheit der heutigen Tools für Performance Engineering oft nur eine eingeschränkte Unterstützung für einige der gängigen Performance Engineering Workflows. Darunter fallen zum Beispiel die Analyse der Skalierbarkeit, der Stabilität und des dynamischen Verhaltens von parallelen Anwendungen, ihre Laufzeitoptimierung und Anpassung für eine bestimmte Architektur oder sogar die Verbesserung ihrer Energieeffizienz um Höchstleistungsrechenzentren zu helfen, die oft sehr hoch anfallenden Betriebskosten zu sinken. Dies, kombiniert mit der hohen Komplexität der heutigen IT-Systemen, macht es extrem schwierig zahlreiche Daten aus Performance-Engineering-Experimente zu sammeln und zu verwalten und gleichzeitig die Entwicklung ihres Codes während des gesamten Software-Entwicklungs-und Tuning-Zyklus fortzuführen.

Solche komplexe, langlaufende Prozesse mit diversen Datenquellen und völlig unterschiedliche Laufzeitumgebungen existieren aber auch in anderen Forschungsbereichen. Beispielhaft, seit vielen Jahren ist das Ziel der Business Intelligence (BI) die Aufbereitung und Auswertung von Daten und Informationen um komplexe Entscheidungen schneller treffen zu können, die Effizienz von Prozessen zu steigern und eine optimale und stabile Arbeitsumgebung bereitzustellen. Nach jahrelanger Forschung wurden viele Werkzeuge entwickelt, um die Verarbeitung von sehr großen Datenmengen zu verbessern und die modellbasierte Entwicklung von Systemen und Komponenten, die den gesamten Analyseprozess automatisieren, zu unterstützen.

Darauf basiert, untersucht diese Dissertation das Konzept der Automatisierung des Performance Engineering Prozesses im Bereich Höchstleistungsrechen (HPC). Sie schlägt ein Framework für die Unterstützung der Anwendungsentwickler bei der Strukturierung und Durchführung des gesamten Leistungsanalyse- und Tuning-Zyklus vor. Gleichzeitig wird auch die Verfolgung der Entwicklung des Quellcodes, der ein Teil des Performance Engineering Prozesses ist, implizit durchgeführt. Das Framework integriert etablierte Standards und Forschungsideen aus dem Bereich Business Intelligence und adaptiert sie für die wissenschaftliche Domäne der parallelen Leistungsanalyse und Tuning von HPC-Anwendungen.

Darüber hinaus diskutiert diese Arbeit neue Erweiterungen des Periscope Performance Engineering Tools, um automatisierte Workflows zu unterstützen und so eine flexible, architekturunabhängige Umgebung bereitzustellen. Als Ergebnis kann ein großes Teil des gesamten Leistungsanalyse- und Tuning-Prozesses automatisiert werden und damit wird auch die Produktivität der Entwickler im Bereich Höchstleistungsrechen deutlich verbessert. Die Auswertung der Arbeit basiert auf einer Reihe von gängigen Performance Engineering Workflows, etablierten HPC Benchmark Codes und andere hochparallele Anwendungen, um die Umsetzbarkeit des vorgeschlagenen Frameworks zu zeigen.

# Contents

Page

**Acknowledgements** iii

**Abstract** v

**List of Figures** xiii

**List of Tables** xvii

**1. Introduction** 1
- 1.1. Motivation and Problem Statement . . . . . 1
- 1.2. Performance Analysis and Tuning Methodology . . . . . 2
- 1.3. Process Automation and Standardization . . . . . 4
- 1.4. Contributions of This Work . . . . . 5
- 1.5. Outline of This Work . . . . . 6

**I. Theoretical Background and Technological Overview** 11

**2. Software Development Life-Cycle** 13
- 2.1. Software Requirements Engineering . . . . . 13
- 2.2. Software Design . . . . . 14
- 2.3. Software Construction . . . . . 15
- 2.4. Software Testing . . . . . 16
- 2.5. Software Maintenance . . . . . 17

**3. Process Automation and Design of Workflows** 19
- 3.1. Foundations of Process Automation . . . . . 20
- 3.2. Process Automation Languages and Standards . . . . . 24
- 3.3. Business Process Management Suites . . . . . 30
- 3.4. Scientific Workflow Automation Tools . . . . . 33

**4. Supportive Software Development Tools** **37**
4.1. Revision Control Systems . . . 37
4.2. Client-Server Repository Model . . . 38
4.3. Distributed Repository Model . . . 41

**5. Related Work** **45**
5.1. The Need for Parallel Programming . . . 46
5.2. Performance Engineering Automation . . . 47
5.3. Performance Engineering Tools . . . 49
5.4. Performance Tuning Libraries and Frameworks . . . 62
5.5. Eclipse and the Parallel Tools Platform . . . 64

**II. PAThWay to Performance Analysis and Tuning Workflows** **69**

**6. Performance Engineering Workflows** **71**
6.1. Performance Engineering Workflows . . . 71
6.2. Workflow for Scalability Analysis of Parallel Applications . . . 72
6.3. Workflow for Cross-Platform Performance Analysis . . . 74
6.4. Workflow for Code Migration and Tuning on GPGPUs . . . 76
6.5. Summary and Requirements Overview . . . 78

**7. Architecture for Automation of Performance Engineering Processes** **79**
7.1. Architecture for Automation of Performance Engineering Processes . . . 79
7.2. PAThWay Graphical User Interface . . . 80
7.3. Logic Implementation Layer . . . 81
7.4. Supportive Modules Layer . . . 83

**8. Workflow Modeling Environment** **85**
8.1. Design Goals . . . 86
8.2. jBPM Workflow Execution Environment . . . 86
8.3. Business Process Model and Notation 2.0 . . . 90
8.4. PAThWay's Custom Domain-Specific BPMN Nodes . . . 98

**9. Internal Data Storage Repository** **113**
9.1. Motivation and Design Goals . . . 113
9.2. Database Entities and Their Application . . . 114
9.3. Object-Relational Mapping using Hibernate . . . 120

**10. Project Documentation Module** **123**
10.1. Motivation and Design Goals . . . 123
10.2. EclipseWiki and Its Features . . . 124
10.3. Integration of EclipseWiki with PAThWay . . . 125

**11. Other Supportive Modules** **129**
11.1. Internal Supportive Modules . . . 129
11.2. Revision Control Module . . . 130
11.3. Environment Detection Module . . . 131
11.4. Runtime Manager and the Parallel Tools Platform Interface . . . 133
11.5. Experiments Browser . . . 136
11.6. Other Graphical User Interfaces . . . 138

**III. Performance Engineering Workflows in Action** **143**

**12. Scalability Analysis Workflow** **145**
12.1. Scalability Analysis Process . . . 145
12.2. Workflow Model . . . 146
12.3. LRZ Linux Cluster . . . 147
12.4. NAS Parallel Benchmarks Multi-Zone . . . 147
12.5. Workflow Execution . . . 148
12.6. Results Exploration . . . 149

**13. Cross-Platform Memory Analysis Workflow** **153**
13.1. Cross-Platform Memory Analysis Process . . . 153
13.2. Workflow Model . . . 154
13.3. STREAM Memory Benchmark . . . 155
13.4. Runtime Environment . . . 156
13.5. Generic Memory Analysis Strategy for Periscope . . . 157
13.6. Workflow Execution . . . 158
13.7. Results Exploration . . . 159

**14. Exhaustive Benchmarking Workflow** **161**
14.1. Exhaustive Benchmarking Process . . . 161
14.2. Workflow Model . . . 162
14.3. SPEC MPI-2007 Benchmarks . . . 164
14.4. Online Data Clustering Support for Periscope . . . 164
14.5. Workflow Execution . . . 166

**15. Generic Performance Tuning Workflow** **169**
15.1. Performance Tuning and Its Application in HPC Centers . . . 169
15.2. Generic Performance Tuning Workflow . . . 170

**16. Conclusion** **177**
16.1. Summary . . . 177
16.2. Future Work . . . 180

**Appendices** **183**

**A. Glossary** **185**

**B. Custom Domain-Specific BPMN Nodes: XML Definitions** **191**
B.1. Application Configuration . . . . . . . . . . 191
B.2. MPI and OpenMP Configuration . . . . . . . . . . 192
B.3. Target HPC System Selection . . . . . . . . . . 192
B.4. Performance Tool Selection . . . . . . . . . . 193
B.5. Source Code Instrumentation . . . . . . . . . . 193
B.6. Experiments Creation . . . . . . . . . . 194
B.7. Experiment Execution . . . . . . . . . . 195
B.8. Runtime Manager Creation . . . . . . . . . . 196
B.9. Load Custom Performance Results to the Database . . . . . . . . . . 197
B.10. Store Additional Information to an Experiment . . . . . . . . . . 198
B.11. Node for Interactive Questions . . . . . . . . . . 199
B.12. Execute a Remote Process . . . . . . . . . . 200

**C. Internal Database Scheme and Data Persistence Classes** **203**
C.1. Scheme of the Internal Database . . . . . . . . . . 203
C.2. Data Persistence Classes . . . . . . . . . . 217

**D. Documentation Module: Modifications to EclipseWiki** **219**
D.1. PAThWay Extension Classes . . . . . . . . . . 219
D.2. Modifications to EclipseWiki's Internal Classes . . . . . . . . . . 220
D.3. Example of a Wiki Markup of a Historical Note . . . . . . . . . . 222

**E. Other Supportive Modules** **223**
E.1. Parallel Tools Platform Interface Module . . . . . . . . . . 223
E.2. Experiments Browser GUI . . . . . . . . . . 225

**F. Bibliography** **227**

# List of Figures

1.1. Performance Analysis and Tuning Methodology . . . . . . . . . . . . . . . . . . . . . 3

3.1. A process model for organizing a business trip based on the Business Process Model and Notation (BPMN) standard . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
3.2. A Petri net of a business process model for organizing a business trip . . . . . . . . . 25
3.3. A UML activity diagram of a business process model for organizing a business trip . 26
3.4. A EPC model of a business process model for organizing a business trip . . . . . . . 27
3.5. A BPMN2 model of a business process model for organizing a business trip . . . . . 28

4.1. Client-Server Version Control Workflow . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
4.2. Distributed Version Control Workflow . . . . . . . . . . . . . . . . . . . . . . . . . . . 41

5.1. Distributed hierarchical architecture of Periscope . . . . . . . . . . . . . . . . . . . . . 50
5.2. Iterative online performance analysis using Periscope . . . . . . . . . . . . . . . . . . . 51
5.3. Overview of the Periscope's GUI for Eclipse . . . . . . . . . . . . . . . . . . . . . . . . 52
5.4. Overview of the CUBE user interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
5.5. Vampir visualizing an event tracing study using Score-P . . . . . . . . . . . . . . . . 56
5.6. Score-P: Architecture and integration overview . . . . . . . . . . . . . . . . . . . . . . 59
5.7. Parallel Tools Platform monitoring a remote system . . . . . . . . . . . . . . . . . . . 66

6.1. Scalability Analysis Workflow Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
6.2. Cross-Platform Analysis Workflow Model . . . . . . . . . . . . . . . . . . . . . . . . . 75
6.3. Workflow Model for Code Migration and Tuning on GPGPUs . . . . . . . . . . . . . 77

7.1. Overview of the architecture of PAThWay . . . . . . . . . . . . . . . . . . . . . . . . . 80

8.1. Original jBPM editor for modeling BPMN workflows using Eclipse . . . . . . . . . . 89
8.2. Enhanced jBPM editor for modeling BPMN 2.0 workflows using Eclipse . . . . . . . 90
8.3. Graphical representation of a BPMN activity . . . . . . . . . . . . . . . . . . . . . . . 91
8.4. Implementation of custom BPMN nodes using the jBPM workflow engine . . . . . . 100
8.5. Custom BPMN node for the selection of an application to be analyzed . . . . . . . . 102

8.6. Custom BPMN node for the configuration of the number of MPI processes and OpenMP threads . . . . . . 103
8.7. Custom BPMN node for the selection of a target HPC system . . . . . . 104
8.8. Custom BPMN node for the selection of a performance tool . . . . . . 105
8.9. Custom BPMN nodes for the creation and execution of performance engineering experiments . . . . . . 107
8.10. Custom BPMN node for the creation of a new runtime manager . . . . . . 108
8.11. Custom BPMN node for instrumenting user's code as part of a performance engineering experiment . . . . . . 109
8.12. Custom BPMN node for storing additional information to an experiment . . . . . . 109
8.13. Custom BPMN node for loading custom performance results from an experiment to the internal database . . . . . . 110
8.14. Custom BPMN node for executing an arbitrary remote process . . . . . . 111
8.15. Custom BPMN node for getting interactive response from the user . . . . . . 112

9.1. Database scheme for storing experiments . . . . . . 115
9.2. Database scheme for storing the configuration of applications . . . . . . 116
9.3. Database scheme for storing the configuration of HPC systems . . . . . . 117
9.4. Database scheme for storing the configuration of performance tools . . . . . . 118
9.5. Database scheme for storing user-specified historical notes . . . . . . 119
9.6. Database scheme for storing performance results generated by Periscope . . . . . . 120
9.7. Object-relational mapping of PAThWay's data classes using Hibernate . . . . . . 121

10.1. An Eclipse view for exploring and creating user-specified notes. . . . . . . 127
10.2. An example of a historical note combining automatically generated information about experiments, user comments and an image with results. . . . . . . 128

11.1. Class structure for tracking user's application code . . . . . . 130
11.2. An overview of PAThWay's experiments browser . . . . . . 137
11.3. User interface for editing available configurations of user's applications . . . . . . 138
11.4. User interface for editing available configurations of performance tools . . . . . . 139
11.5. The first tab of the GUI showing general information about the given HPC system. . . 140
11.6. The second tab showing additional information about the system. . . . . . . 141

12.1. Scalability analysis workflow model . . . . . . 146
12.2. Scalability of BT-MZ from NPB Multi-Zone . . . . . . 150

13.1. Cross-platform memory analysis workflow model . . . . . . 155
13.2. Comparison of the effective memory bandwidth of SuperMUC, LRZ's linux cluster and JuRoPA high-performance systems . . . . . . 160

14.1. Exhaustive benchmarking workflow model . . . . . . 163

15.1. Top-level overview of the performance tuning workflow . . . . . . 170
15.2. Overview of the kick-off sub-process . . . . . . 171

15.3. Overview of the baselining sub-process . . . . . . . . . . . . . . . . . . . . . . . . . . 173
15.4. Overview of the tuning cycle sub-process . . . . . . . . . . . . . . . . . . . . . . . . . 175

C.1. Structure of the Experiment table . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 203
C.2. Structure of the Application table . . . . . . . . . . . . . . . . . . . . . . . . . . . . 206
C.3. Structure of the HPCSystems table . . . . . . . . . . . . . . . . . . . . . . . . . . . . 208
C.4. Structure of the Tools table . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 211
C.5. Structure of the HistoricalNotes table . . . . . . . . . . . . . . . . . . . . . . . . . . 212
C.6. Structure of the PscProperty, PscRegion and PscPropAddInfo tables . . . . . . . . . . 213
C.7. Scheme of PAThWay's internal database . . . . . . . . . . . . . . . . . . . . . . . . . 216
C.8. Data Persistence Classes (ORM) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 218

D.1. Structure of the PAThWayTextRegion class . . . . . . . . . . . . . . . . . . . . . . . . 219
D.2. Structure of the PAThWayTextRegionMatcher class . . . . . . . . . . . . . . . . . . . . 220
D.3. Structure of the TextRegionVisitor classes . . . . . . . . . . . . . . . . . . . . . . . . 220
D.4. Structure of the LinkMaker classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . 221
D.5. Structure of the WikiLinkLauncher class . . . . . . . . . . . . . . . . . . . . . . . . . 221

E.1. User interface for showing all runtime environment variables . . . . . . . . . . . . . 225
E.2. User interface for showing the loaded external modules . . . . . . . . . . . . . . . . . 225
E.3. User interface for showing the standard output log . . . . . . . . . . . . . . . . . . . 226
E.4. User interface for showing the compilation log after (re-) instrumentation . . . . . . 226

# List of Tables

8.1. An overview of activity types and markers supported by BPMN 2.0 . . . . . . . . . . 93
8.2. An overview of all events supported by BPMN 2.0 . . . . . . . . . . . . . . . . . . . . . . 95
8.3. An overview of the gateway nodes supported by BPMN 2.0 . . . . . . . . . . . . . . 97

12.1. NAS Parallel Benchmark Multi-Zone Problem Classes . . . . . . . . . . . . . . . . . . . 148
12.2. Scalability Results of NPB BT-MZ Class C . . . . . . . . . . . . . . . . . . . . . . . . . . . 151

13.1. STREAM benchmark: synthetic kernels . . . . . . . . . . . . . . . . . . . . . . . . . . . . 156

# Chapter 1

# Introduction

## 1.1. Motivation and Problem Statement

Tools for parallel performance analysis have been the main topic of research for many institutions for already more than 20 years. However, parallel application developers are still complaining about lacking tools for intuitive performance analysis. In her paper from 1999 [165], Cherri Pancake described the state of such tools as unusable by the user community at that time. The information provided by the majority of these tools was either too overwhelming for the developers or its presentation was simply inappropriate. Moreover, performance tools were thought of being too hard to learn and use effectively. As a consequence, many developers were unwilling to invest time in learning to use performance analysis tools mainly due to the reason that a successful outcome was not guaranteed.

Nowadays, more than 10 years after the publication of Pancake's paper, the acceptance of tools for parallel performance analysis has definitely improved but not to the extend it should ideally be. Even though the design and functionality of performance analysis tools have been greatly enhanced, such tools are still not widely applied. The main complain of the users community continues to be that the majority of performance tools are still too hard to learn or they do not support the tasks users commonly have to perform while trying to optimize their applications.

What is more, the mainstream performance analysis tools do not support doing some very common tasks like scalability analysis and testing different execution configurations. The developers are expected to manually apply these tools many times in order to get statistics about their applications. Besides that, these tools often provide very limited functionality for the comparison of different runs and, thus, it is extremely difficult for the users to analyze and organize the collected performance data.

It has to be mentioned, however, that problems such as processing large amount of data, diverse data sources and totally different environments also exist in other research areas. Making relevant data-based decisions that lead to improving process efficiency has been the goal of business intelligence (BI) for many years now. A lot of research has been done in this field in order to accelerate and improve the overall decision process and create a smooth and stable working environment. Moreover, many tools have been developed to support the analysis of huge data quantities and create model-driven, easier to use systems.

If the existing BI tools are closely studied, it is straightforward to see that they all support one common but very powerful feature: control and data workflows. This aspect of BI tools has been proven through the years to be able to encapsulate the great complexity of multiple data sources coming from diverse technologies and greatly automate the overall process.

In the field of high-performance computing, workflow tools have not seen a broad acceptance up to now. This is, however, mainly due to the fact that they have either been too specific to the requirements of a particular group or had the wrong targets for automation. Tools utilizing workflows can have a significant impact in the area of parallel performance analysis and optimization. The repetitive nature of this practice makes it a perfect target for automation. Such a system can, for example, utilize workflows for analyzing the scalability and stability of parallel applications, studying their dynamic behavior, optimizing them for a particular architecture or even improving their power efficiency to lower the operational costs of HPC centers.

Why should the HPC community undergo the same mistakes done in other fields and not simply learn from their experience? Why loosing time in manually running different performance experiments and managing the resulting data and not automate the whole process using high-level HPC workflows?

## 1.2. Performance Analysis and Tuning Methodology

In the field of high-performance computing maintenance, as defined in Chapter 2.5, is one of the most common and lengthy phases of the software development life-cycle. It is often motivated by the excessive costs of running ineffective parallel applications on big parallel machines. However, optimization of scientific simulations, i.e. the most common applications on HPC systems, is mainly pushed by the eager to run longer and more precise simulations in order to get a better understanding of the underlying research question.

Although the performance of applications can be optimized based on programmer's sixth sense, it is normally not the most productive approach one can take. This technique is even more counterproductive when it comes to the tremendously complex parallel computing resources nowadays.

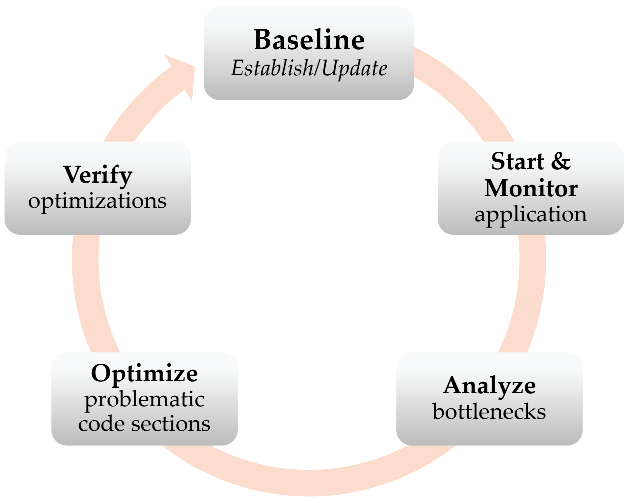


Figure 1.1.: Performance Analysis and Tuning Methodology

A better approach is to follow a specific methodology similarly to the one commonly adopted in the standard software development process (see Chapter 2). On an abstract level, the performance analysis and tuning process, or also commonly called performance engineering, can be roughly split into two tightly connected tasks: performance analysis and performance tuning. These two phases typically form an iterative process, as shown in Figure 1.1, during which the user usually goes through the following sequence of steps:

**Baseline** Before starting a new tuning iteration, a set of performance optimization objectives has to be specified. This set has to specify the desired improvement goals in terms of memory usage and/or execution time. Additionally, tests have to be chosen that can confirm or reject a specific source code modification with respect to the specified goals. A software revision control system can be applied to track the code changes and easily create/update the baseline.

**Start & Monitor** This step involves compiling of the target source code, selecting a proper runtime configuration and starting the application on the parallel system. This step might also include multiple simultaneous executions using the same execution configuration.

As part of this step, the execution of the application has to be also closely monitored using, e.g., hardware counters, library wrappers, specific source code instrumentation, etc. This task is normally done using specialized performance analysis tools which collect data about the runtime behavior of the monitored application. This process is also commonly called a performance analysis experiment.

**Analyze** After the experiment completes, the performance data have to be analyzed to detect potential problems, also known as bottlenecks. This step of the cycle is often the hardest as it typically requires a deep knowledge about the underlying hardware architecture and the network topology in order to understand if there is room for improvement. There exist different tools for automating this step: some of them provide a plain list of hardware counters and their values, while others go one step further by preprocessing and evaluating the raw performance data with the aim to create high-level performance properties. Taking into account the high complexity of today's and future hardware platforms, the second group of performance analysis tools provide a clear advantage for the developer. However, one must often use many of them simultaneously to get the whole picture.

**Optimize** When the performance properties are analyzed and the bottlenecks are identified, the necessary tuning steps have to be defined. This can range from small source code modifications to the replacement of whole algorithms.

**Verify** After the proposed optimization strategy is implemented, it has to be verified that the modifications really improve the performance. This is typically done by repeating the Execute, Monitor, and Analyze steps. If the performance improvement can be successfully proven, the optimization has to be recorded, the software baseline updated and the tuning cycle can then start a new iteration in searching for new optimization opportunities. However, if there is no real improvement in the runtime behavior, the modifications have to be undone and the process has to be restarted at the Optimize step. Due to its intrinsic complexity, this step requires that one has a full backlog of all previous steps and decisions in order to choose the appropriate restart point.

In the last years, each step of the tuning cycle was thoroughly researched. As a consequence, the monitoring, analysis and optimization steps are now reasonably advanced and provide a varying degree of automation. However, there is not so much research about how to combine existing performance tools and automate the presented cycle in order to address the complexity of HPC systems and enhance the tedious process of performance engineering.

## 1.3. Process Automation and Standardization

With the vast advancement of technology and the broad adoption of computer systems by enterprises, a lot of the regular business processes are nowadays fully automatized. As discussed in Chapter 3, these include a wide set of activities ranging from simple planning of raw materials over tracking and controlling supply and logistics to even maintaining customer relations. By automatically applying predefined best practices and enforcing existing compliance regulations without having to rely on human interactions, companies can greatly enhance their productivity and boost their revenues. That is why, a separate field of research and development has been established worldwide to investigate different ways of standardizing the modeling and the automation of business processes. As a result, diverse specification languages and complex software systems have emerged to provide the means necessary for creating highly automatic business execution environments.

This trend can also be identified in the field of information technology. For example, the Information Technology Infrastructure Library (ITIL) [132] specifies a collection of best practices for IT service management in the form of process descriptions. This way, the inherent risks related with IT operations can be lowered while simultaneously increasing the operational efficiency of companies. Thus, adopting a higher degree of workflow automation and standardization proves to be also a very good long-term strategy for the majority of IT-related companies.

However, this strategy is generally not applied in the fields of high-performance computing and performance engineering. Despite the fact that most of the processes in these areas are highly iterative and thus could directly benefit for a higher-degree of automation, these are normally performed manually by the involved users without the support of software systems.

What is more, even though the workflows used by the majority of the HPC developers, i.e. scalability analysis, cross-platform benchmarking, performance tuning and others, are very similar in nature, there is no common set of best-practices shared by the community or any other form of standardization. This inevitably leads to highly inefficient processes, bigger operational expenses for both supercomputing centers and HPC developers and unpredictable overall quality and outcome for the related projects.

## 1.4. Contributions of This Work

First and foremost, this dissertation presents PAThWay: a new software system for designing, executing and monitoring performance engineering processes and tracking the evolution of the source code throughout the overall development and tuning cycle which is commonly found in the area of high-performance computing. The framework, the name of which is an acronym for Performance Analysis and Tuning Workflows, uses a palette of technologies like the Business Process Model and Notation (BPMN), a relational database management system (RDBMS), wiki-based documentation, source code revision control and others. It aims at enhancing the performance analysis and tuning process by abstracting its inherent complexity and thus tries to improve productivity of HPC developers. Furthermore, the framework provides a set of predefined workflows of common performance engineering processes which can serve as a starting point for designing more complex processes or, for example, for helping inexperienced developers to enter the field of HPC by providing them with a set of best practices.

Moreover, being developed in the context of the Periscope [22] automatic performance analysis tool, PAThWay can take advantage of the newest developments in the area of performance analysis and automatic tuning. This not only allows the framework to provide a flexible environment for automation of activities such as scalability analysis, manual or automatic optimization cycles, parameter studies, etc., but can also interface with existing cross-platform comparison tools like Cube [189] and PerfExplorer [87] for the sake of displaying tuning progress and post-processing collected performance data.

Furthermore, this thesis proposes a complete end-to-end workflow for the automation of one of the most crucial operational processes in supercomputing centers: the performance tuning of user's applications. It includes both a high-level overview of activities in a typical runtime optimization project and a precise definition of the separate iterative processes that are part of it such as the baselining phase and the long-running tuning cycle presented in the previous section. Additionally, an approach for building a set of best practices for tuning applications is recommended as part of the workflow (see Chapter 15). On the longer term, these ideas together with the PAThWay framework try to provide a baseline for standardizing and automating not only the performance tuning process but also eventually the rest of the daily HPC activities as already done for the field of IT service management.

Last but not least, this dissertation discusses the motivation behind two new features of the Periscope performance engineering toolkit: an automatic, cross-platform analysis strategy for identifying memory bottlenecks and an online, multivariate statistical clustering functionality for summarizing the performance data gathered from large-scale experiments.

## 1.5. Outline of This Work

The reminder of this dissertation is organized in three parts. The first part presents the theory behind process automation and discusses the typical software development and performance engineering activities. Furthermore, it introduces the work of others that has direct impact on PAThWay and the ideas presented in this dissertation. This brief theoretical and technological overview is structured as follows:

**Chapter 2** presents the standard software development life-cycle as specified by the Institute of

Electrical and Electronics Engineers (IEEE) and discusses its separate phases, i.e. requirements engineering, design, construction, testing and the consecutive maintenance.

**Chapter 3** recalls the theoretical foundations required for both understanding existing workflow models and designing new ones. It starts by defining common terms used in the field of process automation and then present the most popular workflow specification languages and standards. Afterwards, some of the well-established process modeling environments from the areas of both business and science are discussed.

**Chapter 4** gives an overview of different types of revision control systems as means of supporting the entire software development life-cycle. It discusses the characteristics of both client-server and distributed repository models and compares their advantages and disadvantages.

**Chapter 5** defines the basics of performance engineering and introduces related scientific work. Furthermore, it elaborates on current state-of-the-art tools for performance analysis and tuning such as Periscope [22], Scalasca [71], TAU [187] and the Score-P [146] runtime monitoring library. Following is an overview of common software frameworks for performance tuning and automatic optimization libraries. Last but not least, this chapter introduces the Eclipse [56] integrated development environment and one of its plug-ins which is specifically designed for supporting the development of parallel applications, i.e. the Parallel Tools Platform (PTP) [222].

Coming after the first part of the dissertation is a detailed discussion of the PAThWay framework and its features. It is organized in the following six chapters:

**Chapter 6** discusses the high-level definitions of three popular performance engineering workflows in order for the reader to get a broader understanding of the involved processes and their requirements. Additionally, an approach for automating them is presented and it is consequently mapped to the motivation behind the development of PAThWay.

**Chapter 7** introduces the PAThWay framework and elaborates on its overall software architecture. It includes a short description of all the major software components together with their interrelations and dependencies.

**Chapter 8** gives a broad overview of the Business Process Model and Notation (BPMN) standard, the jBPM [200] workflow execution engine and the resulting workflow modeling environment that comes with PAThWay. This chapter is then extended by a discussion of special

BPMN activity nodes that were developed in order to customize the workflow engine to the requirements imposed by a typical performance engineering process.

**Chapter 9** presents the internal data storage repository of PAThWay and explains its design goals and role within the overall architecture for automation of performance engineering processes.

**Chapter 10** brings up the topic about the importance of having an accessible documentation about requirements, work progress and ideas on the success of software projects and proposes a documentation module that can be used together with the workflow environment.

**Chapter 11** describes some of the internal software components of PAThWay that contribute the creation of the overall automation environment. Moreover, this chapter includes an outline of graphical user interfaces for configuring and customizing the framework and providing the access to the collected performance engineering data.

The third part of this work presents three different performance engineering workflows that were completely implemented and tested with PAThWay. Additionally, a more complex, multi-step workflow model for automating the typical performance tuning cycle is presented together with a discussion of its activities. The part is logically structured as follows:

**Chapter 12** presents the requirements and the process model of a workflow for performing scalability analyses and provides a description of its implementation within PAThWay. Additionally, it elaborates on the execution of this workflow using the NAS Parallel Benchmarks Multi-Zone suite [108] and shortly discusses the collected performance analysis results about the scalability of the BT-MZ benchmark.

**Chapter 13** introduces another common performance engineering workflow that aims at automating a cross-platform memory analysis process. Similarly to the previous chapter, a discussion of the model and its specific implementation is included. The workflow is then tested using three different HPC systems and the well-established STREAM [144] benchmark. In addition to a discussion of the gathered results, the new generic memory analysis strategy for Periscope is also presented.

**Chapter 14** discusses the exhaustive benchmarking performance engineering process that is often applied by hardware vendors and technical consultants. Additionally, it demonstrates the workflow model of this process which was designed with PAThWay and evaluated using the weather research and forecasting code called `127.wrf2` [148] from the well-known

SPEC MPI-2007 [152] benchmarks. Additionally, the online clustering functionality of Periscope is introduced as an effective way of aggregating the data collected from performance engineering experiments such as the exhaustive benchmarking process.

**Chapter 15** defines the end-to-end performance tuning workflow and gives a motivation on how it can be applied in the real-world situations to enhance the overall tuning process. As a result, the quality of the delivered optimization results can be eventually improved and thus the success rate of performance engineering projects can be consequently increased.

Finally, this dissertation concludes in **Chapter 16** with a short summary of the work and a discussion of the future directions for PAThWay.

# Part I.

# Theoretical Background and Technological Overview

Chapter 2

# Software Development Life-Cycle

**Contents**

**2.1. Software Requirements Engineering** . . . . . . . . . . . . . . . . . . . . . . . . . 13
**2.2. Software Design** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
**2.3. Software Construction** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
**2.4. Software Testing** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
**2.5. Software Maintenance** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

## 2.1. Software Requirements Engineering

Even though there are many different software development life-cycle models [167], the typical process of creating new products always begins with the extraction of customer's requirements. There are two major types of software requirements: product and process ones. The first type deals with specifying the goals of a software and its externally visible behavior. These are typically high-level definitions and come directly from the stakeholders. The second type of software requirements defines the implementation restrictions such as programming language, fault-tolerance support, process auditing policies, etc. As specified in the IEEE standard for requirements engineering [105], these can be explicitly specified from the customer or they can be implicitly created by other product requirements. Furthermore, the software requirements can be separated based on their effect on the product in functional and non-functional. The first ones specify the features of the final product and are often much easier to verify later on. The non-functional requirements have direct implication on the development and the successive execution of the software and are commonly regarded as quality constrains such as performance requirements in terms of response time and scalability or fault-tolerance behavior.

The requirements engineering process is not a one time task that is always executed in the beginning of a new software development project but rather an iterative activity that exists throughout the whole life-cycle of the product. As discussed in the software requirements book of Robertson [176], this process can be subdivided in the following phases:

- **Elicitation** - this step aims at identifying the stakeholders and capturing the initial software requirements. This is an activity that usually involves multiple observations, interviews and meetings with stakeholders, designing of common use-cases and creating simple prototypes in order to acquire all relevant requirements.

- **Analysis** - during this phase high-level models [100, 99] of the software and its execution environment are developed using the Unified Modeling Language (UML) [179] . This activity targets the identification and resolution of conflicting requirements and the specification of project boundaries. Moreover, the future software architecture together with its requirements is discussed.

- **Specification** - after a precise analysis of the requirements, they have to be formally defined. This phase deals with the creation of up to three formal documents describing all software requirements: Concept of Operations Document (ConOps) [97], System Requirements Specification (SyRS) [96] and Software Requirements Specification (SRS) [98]. The first one describes the whole system from the perspective of a domain-expert and does not contain any low-level implementation details. The SyRS is created only for products that combine the development of both hardware and software components and specifies the non-software related requirements. The SRS is the most fundamental document from software engineering point of view as it defines all software constrains and their internal relations.

- **Validation** - the last phase of the requirements engineering process deals with the review and the verification of all specification documents according to stakeholder's requirements about the future software [102].

## 2.2. Software Design

Design is the second phase of the software life cycle. While the requirements phase defines the problem in the form of goals, constraints, and solutions, the design phase concentrates on choosing the future architecture of the software by evaluating alternative solutions that fit at most to all software requirements, target execution environment and project's schedule and resources. Furthermore, software engineers have to carefully examine the specification documents from the previous phase in order to define the software as a set of finite number of highly detailed, interconnected components. These modules are often created using design patterns that are based on the best practices for solving different software problem [32]. Moreover, the interfaces between all components have to be precisely defined as part of this stage.

The software design phase can greatly differ from one field to another. For example, the great diversity of high-performance computing systems, their inherent complexity, and the extremely fast hardware evolution with very short product life cycles, make the design of HPC applications often one magnitude more difficult in comparison to typical software development. Software engineers have to not only select a specific software architecture, but they also have to analyze its compatibility to the underlying hardware before a final design could be chosen.

## 2.3. Software Construction

Directly following the design is the software construction phase. During this step, the chosen architecture and its components have to be implemented and then they have to be extensively tested against all requirements. This is often the most work-intensive phase and requires a high level of expertise from the software engineers. Depending on the chosen software development life cycle (SDLC) model, the software construction can be a single long-running process, e.g. in waterfall [177] and feature-driven development [162] models, or it can be formed from many a short incremental iterations like in rational unified process [124], extreme programming [20] and Scrum [183]. A short comparison between the most popular SDLC models can also be found in [5].

During the construction phase, there are three major groups of languages that can be used to create a running solution:

- **Configuration languages** are the simplest among the three types and are commonly used by business-related professionals rather than software engineers. Often, customizing the behavior of already existing software products by adapting their configuration is also the most cost effective approach for creating new software assets.

- **Toolkit languages** are domain-specific specifications provided by different software frameworks for the creation of new applications. Additionally, customizable templates are usually supplied. They provide more flexibility in the implementation of required software that the first group but are still much easier to use that the programming languages in the third group.

- **Programming languages** provide the most general way a solution can be implemented. However, they also require much more expertise from the software engineers and are generally the most expensive option.

Another major step in the software construction phase is the testing of both the intermediate and the final solution. It typically comprises of unit tests [103], i.e. the analysis of the behavior of the separate software components against their requirements, and investigation of the proper integration of all components within the solution and the target execution environment.

## 2.4. Software Testing

The last phase before the official release of a software product deals with its extensive testing. In the past, this step was executed only after the software construction was finished. However, due to the increasing complexity of products and the tight project schedules, it now spans over both the construction and maintenance phases. Sometimes it can even precede the actual software development as in the test-driven development (TDD) [19].

Depending on the knowledge about the system under test, there are black-box and white-box testing techniques. As discussed in [1], the generation of test configurations can be based on developer's experience, on requirements specifications, on code analysis techniques or on use-case scenarios. Furthermore, the different software techniques can be subdivided according to their goals in the following groups:

- Conformance testing - verifies that all functional requirements of the product are met.
- Performance and stress testing - analyzes the non-functional requirements and the reliability of the software.
- Installation and configuration testing - investigates the installation of the software in the target environment and its dynamic behavior using different configuration settings.
- Recovery testing - analyzes the ability of the system to automatically recover after a major failure.
- Alpha and beta testing - autonomous testing of the complete software by a small, precisely selected subset of the future users. These so called early adopters are part of the software development organization in case of alpha tests or are external parties during beta testing.
- Back-to-back testing - compares the results of running the same test case using two different versions of the product.

- Regression testing - similar to the back-to-back testing but uses more test cases and aims at identifying faults between different software versions as early as possible.
- Usability and acceptance testing - analyzes if the requirements of stakeholders are met and how easy it is for the new users to learn and operate the constructed software.

Due to the high complexity of this phase and the common inability of developers to identify defects in their own code, software testing is often done by an external team of test engineers or by people not directly involved in the development of the software. Moreover, this also requires that the process is precisely planned and closely managed in order to avoid excessive costs due to too extensive testing.

## 2.5. Software Maintenance

After releasing a software product, it normally enters the stage of maintenance. This phase is not only concerned with fixing defects but also with performance improvement of the software, adaptation to new hardware systems and implementation of new feature enhancements. According to the official IEEE Standard for Software Maintenance [101] and its revision in 2006 [104], there are four types of software maintenance:

- Corrective - targets the repairing of software defects after they get reported by end-users;
- Adaptive - modification of the software in order to support new execution environments;
- Perfective - activity initiated by the maintainers having the goal to optimize the performance behavior and the maintainability of the software;
- Preventive - proactive maintenance type that aims at eliminating possible software defects before they are encountered by the end-users;

The phase of software maintenance can be the longest one spanning over many years. Moreover, it is commonly performed by separate teams or even specialized companies that were not involved in the initial development of the software. These factors combined with the predominantly lacking communication between software development and maintenance teams create a very complex working environment. Furthermore, this phase of the software life cycle is often ignored by managers since no new products are created and it has no direct effect on achieving the business goals of a company. As a result, its execution is usually not properly planned which eventually leads to very high costs for both the development organization and the end-users.

Chapter 3

# Process Automation and Design of Workflows

**Contents**

**3.1. Foundations of Process Automation . . . . . . . . . . . . . . . . . . . . . . . . . . . 20**

**3.2. Process Automation Languages and Standards . . . . . . . . . . . . . . . . . . . . . 24**

3.2.1. Petri Nets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

3.2.2. Unified Modeling Language Activity Diagrams (UML AD) . . . . . . . . . . 26

3.2.3. Event-driven Process Chains (EPC) . . . . . . . . . . . . . . . . . . . . . . . . 26

3.2.4. Business Process Model and Notation (BPMN) . . . . . . . . . . . . . . . . . 27

3.2.5. Other XML-based Languages for Process Modeling . . . . . . . . . . . . . . 29

**3.3. Business Process Management Suites . . . . . . . . . . . . . . . . . . . . . . . . . 30**

3.3.1. IBM Business Process Manager . . . . . . . . . . . . . . . . . . . . . . . . . . 30

3.3.2. Appian BPM . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

3.3.3. JBoss Drools and jBPM . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

**3.4. Scientific Workflow Automation Tools . . . . . . . . . . . . . . . . . . . . . . . . . 33**

3.4.1. Kepler . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

3.4.2. Taverna . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

3.4.3. Pegasus . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

3.4.4. Domain-Specific Automation Languages . . . . . . . . . . . . . . . . . . . . . 35

## 3.1. Foundations of Process Automation

The field of process automation has a very long history dating back to the 18th century. At that time, it included mainly manual automation where bigger manufacturing processes were split in smaller, independent tasks that can be performed by highly specialized workers. The employees had to learn only a single, easy to handle task and did not need know the whole process chain. The advantages of that approach for the companies were the much lower costs for production and human resources due to the simplicity of the tasks and the higher degree of automation. With the further development of technology, this trend kept increasing and eventually all those manual tasks were replaced by a network of highly efficient robots. Many enterprise functions ranging from planning of required raw resources over tracking and controlling their supply and logistics to even maintaining customer relations were automatized using complex software systems. As a result, the time to market of new products was severely decreased and the manufacturing costs were successfully scaled down, thus opening completely new opportunities for the companies worldwide.

Due to the inherent complexity of this widespread migration to automated processing and the vast amount of involved systems, the creation of some sort of standardization became crucial for the successful continuation of the trend. Consequently, a new field of research and development was created which unites both the business administration and the computer science communities. The main idea behind this business process management field is to provide a structured way to design, execute, monitor and optimize different activities with the aim of achieving certain goals [211]. Having its origins in the area of enterprise management and organization, a major part of its terminology comes from the business administration area with some adaptation to better map it to the underlying software systems. The most fundamental logical unit of business process management is a business process. It was initially defined as a collection of activities that require certain inputs and produce outputs that are of interest for a particular customer [84]. However, this definition proved to be too vague for structuring today's complex processes and, as defined in 3.1, it was eventually refined to include additional information about the specific environment and the internal relations between the involved activities [52]. Furthermore, even though business processes can be used to describe tasks that span across the boundaries of a given organization (a design technique also known as *process choreography* [35]), they always have only a single owner. This constrain allows the detailed analysis and optimization of the processes with respect to the responsible organization which is one of the fundamental task of business administration professionals.

**Definition 3.1.** A *Business Process* is a set of correlated activities conducted in a predefined environment (technical or organizational) having the common aim to achieve a particular business goal such as provide a service or release a product [223].

As previously mentioned, the field of business process management developed as a consequence of the efforts to automatize the intrinsic processes of companies. However, since most of these steps are nowadays based on software systems, a more standardized description of business processes was developed so that the resulting processes can be autonomously carried out by existing IT systems. These process models, as defined in 3.2, not only specify manual and automatic activities of a process as an ordered set of nodes, but also impose additional constrains on their execution such as conditional branches and possible error states. Just like classes in object-oriented programming (OOP) , business process models lay the foundations and the behavior of processes. At runtime, a unique business process instance has to be created from a model in order to store its state and be able to conduct the required activities. For the definitions of such models, there exist many other specifications and design languages. A subset of them will be discussed in further details in Section 3.2.

**Definition 3.2.** A *Business Process Model* is a formal, application-independent description of a set of activities, their interrelation and any existing execution constrains among them that are needed for the successful realization of a business process [7]. A *workflow model* is another commonly used synonym of a process model which is more common in the scientific and computer science communities.

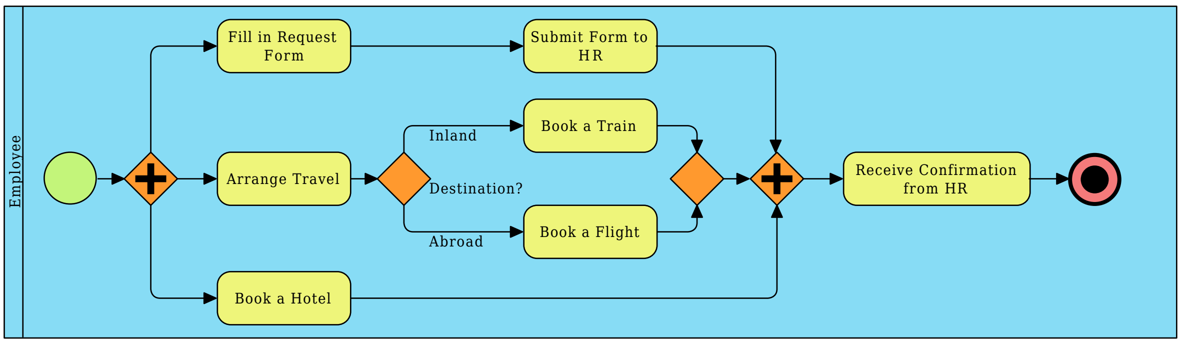


Figure 3.1.: A process model for organizing a business trip based on the Business Process Model and Notation (BPMN) standard

In order to better illustrate the idea behind business process models, the activities done by an employee to organize a business trip are shown in Figure 3.1. The process typically consists of the following steps:

- Filling in an official form for requesting allowance for a business trip.
- Submitting the filled-in form to the human resources (HR) department.
- Making travel arrangements according to company's regulations. Depending on the destination of the business trip, there are normally different transportation means allowed:
  - Train must be chosen for all domestic destinations.
  - A flight can be booked for all abroad destinations.
- Book a hotel at the target destination.
- Wait for the receipt of the official allowance to go on a business trip.

The presented model is based on the established Business Process Model and Notation (BPMN) [7] standard which includes both textual representation of the model using the XML format and a graphical one (see Chapter 3.2.4). The example contains only a small subset of the provided design entities. The start of the process is manually triggered by the employee and is shown in the figure using an empty circle. The separate activities that are part of the process are depicted as connected rectangles where the name of the task is given in the middle. The arrows in the model represent execution constrains and diamond shapes are used to describe conditional split and join gateways. For example, the form has to be first filled in by the employee before it can be send to the HR department.

However, making travel arrangements and booking a hotel at the destination are tasks that do not necessary depend on the submission of the official form and so can be done in parallel. In the model, parallel branches are created and later joined back together by an *and/parallel split* gateway drawn using a diamond shape with a plus sign in it.

Furthermore, in many companies there are normally special requirements on type of transportation that can be chosen depending on the final destination. This condition is depicted in the presented model using a diamond shape which is either empty or has a diagonal cross/saltire in it. It stays for an *exclusive or/split* gateway which means that only one branch can get executed at a time: the employee can either book a train ticker or a flight. The chosen branch is then merged back to the standard flow of the process using another empty diamond shape.

After all parallel branches complete, i.e. filling in the form, arranging the transportation means and booking a hotel, the employee has to wait for the official allowance from the HR department. After its reception, the process terminates (shown using a filled circle with a thick border) and the business trip can officially begin.

After covering the concept of business processes and their formal representation, the available means for effectively implementing those process models have to be discussed. Even though a business process model can be highly detailed, its main idea is to represent the underlying activities and their correlations and may not provide information about the specific execution of the model on a given software system. Therefore, a concept from the field of computer science is commonly used to complement business process models - automation modeling using workflows. In the past workflows mainly described the steps a software system had to complete in order to achieve a certain goal. In addition, the data that were exchanged between the separate steps had to be explicitly defined. However, in contrast to business process models, workflows used to be very technical and often contained no high level documentation about the underlying tasks and their dependencies. However, the recent developments of the business process modeling field and the release of the second version of the BPMN standard, the differences between business process models and workflows are practically not existent any more.

**Definition 3.3.** A *Workflow Management System (WFMS)* is a software tool that provides different execution engines for the automatic routing of data and tasks for further processing among participating entities in a specific business process. This can involve triggering of manual activities or automated functions. However, integration with external systems is typically limited [223].

Following the trend for automating today's enterprises, specialized software systems were developed for the execution of business process models. Currently, there exist two major types of such systems: the much simpler workflow management systems (see Definition 3.3 and [74]) and the more complex business process management suites (BPMS). Though they both have the same main goal of automating common processes, there is a major difference between them. In comparison to workflow management systems, a BPMS targets bigger, enterprise-wide projects and focuses on the product life-cycle management of business processes instead of the implementation of the pure data and tasks exchange. Furthermore, as defined in 3.4, BPMS is considered a superset of the workflow systems which includes additional components such as central repositories with process models and templates, different user roles, content and change management functionality and many others [210, 122, 223]. Another major difference between the two types of process automation systems is that a BPMS has support for offline simulation and optimization of business process models. As a result, execution bottlenecks can be more easily detected and the formal business processes models can be improved in order for a company to achieve higher operational efficiency and thus get a competitive advantage.

**Definition 3.4.** A *Business Process Management Suite (BPMS)* is a specialized software system for defining, managing, executing and optimizing business processes independently from a specific application. Its foundations are based on a workflow management system. However, it provides additionally modules for managing, auditing and reporting of process instances; optimizing existing business processes; and seamless enterprise application integration (EAI) support [7].

After covering the most important principles of process automation and business process management, the next chapter will give an overview of the most popular specifications for designing formal models of processes. In Chapter 3.3, the state-of-the-art tools for business process management will be covered followed by the automation tools commonly applied in the field of high-performance computing in Chapter 3.4. This overview is further complemented by a discussion of domain-specific languages for modeling repetitive scientific experiments.

## 3.2. Process Automation Languages and Standards

In the course of time, many different languages and standards for describing business processes have emerged. Some of them provide pure text-based XML specifications with no graphical notations. Others like the Unified Modeling Language support graphical modeling but have not seen broad acceptance by the community. Moreover, process models created using some specifications can be carried off autonomously by software systems while others have to be first converted in different formats to allow automatic execution. An short overview of the most popular standards for business process modeling will be given in the following chapters to better illustrate this immense diversity.

### 3.2.1. Petri Nets

Petri nets [155] are one of the earliest specifications for designing and simulating process workflows [209] and complex concurrent systems [73]. They provide both a formal mathematical definition of the modeled entities and an abstract graphical notation. However, in their original version petri nets do not support the description of the runtime environment or other high level properties commonly found in business processes.

Using this specification, both the static and the dynamic state of processes can be modeled. The first one is designed using three main components: *places, transitions* and *directed arrows* that connect them. The places describe passive entities of the process such as conditions or states and are graphically represented using circles. The work items are defined using transitions and are displayed as rectangles with the names of the activities inside them. A key property of petri nets is that they are bipartite graphs, i.e. no direct connection between two transitions or two places can exist. In other words, each place in the model is always followed with a transition that actively changes the state of the process (see Figure 3.2). The places preceding a transition are called the input places of that transition and the ones following are its output places.

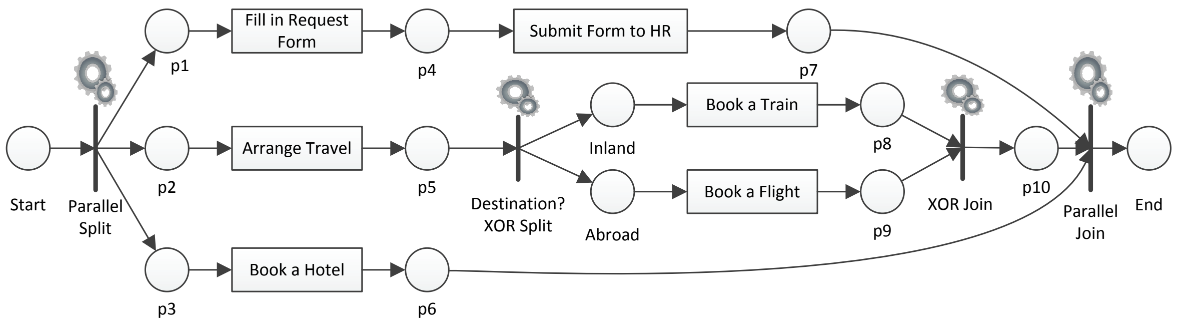


Figure 3.2.: A Petri net of a business process model for organizing a business trip

The dynamic behavior of processes, i.e. their process instances, is presented using tokens that occupy the places in a static model. These tokens are actively moved between the different places using the transitions which is also known as a *token play*. The original version of the specification says that a transition between two places can only occur if all input places contain a token. This restriction together with the fact that tokens cannot carry any data restricts the application of petri nets to real-world problems. As a result, three different extensions were developed in the course of time: *condition event nets*, *place transition nets* and *colored petri nets*.

The condition event nets restrict the number of simultaneous tokens in a particular place to only one and define the places as conditions for the tokens, i.e. a token residing in a place always fulfills the condition imposed to it by that place. Transitions in a conditional event net get activated only if all input places contain a token and all the output places are empty. Just like the standard petri nets, the tokens in such nets do not carry any additional information.

Another extension of the petri nets specification are the place transition nets. The difference here is that any number of tokens are allowed to reside concurrently in a particular place. As a result, the places in these nets are commonly used as counters for the passing through tokens and not to insure that a specific condition is met. The transitions are enabled when a certain number of tokens arrive in the associated input places.

The last and most useful development of petri nets are the colored petri nets. They specify that each token in the model always carry values of particular types. These types define the range of values a token can contain and are also known as color sets. In addition to the features from the previous two net types, the transitions in a colored net get activated when the data in the input tokens matches a particular condition. Moreover, the directed arrows can impose supplementary conditions on the activation of transitions again based on the data carried by a token.

### 3.2.2. Unified Modeling Language Activity Diagrams (UML AD)

Activity diagrams [58] are part of the Unified Modeling Language (UML) [179] and provide a graphical notation for process models. In the first version of the UML standard, activity diagrams were used to depict only the static state of processes and they had no formal specification. However, with the release of UML 2.x, the semantics of activity diagrams was extended based on petri nets and now they include additional support for modeling the dynamic behavior of processes [194].

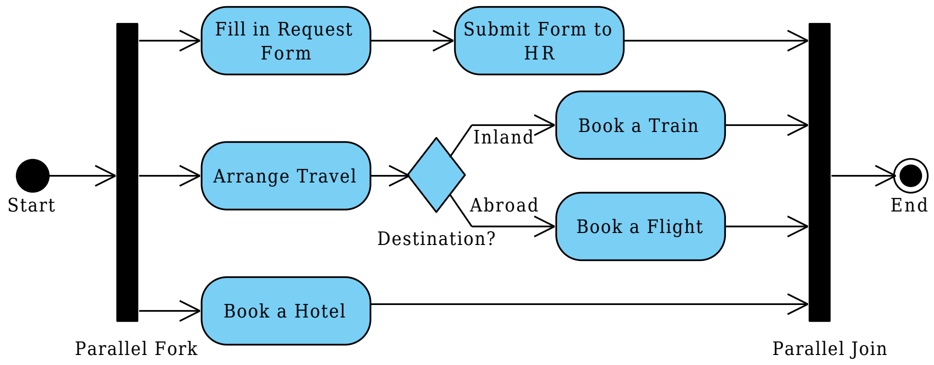


Figure 3.3.: A UML activity diagram of a business process model for organizing a business trip

In order to better illustrate their application, a UML activity diagram of the sample business process of requesting a business trip from Chapter 3.1 is shown in Figure 3.3. As specified in the standard, the initial state of a process is depicted using a filled circle. Another filled circle with an additional border is used to signal the termination of the sequence of tasks. All activities of a process are displayed using rounded rectangles with the names of the tasks inside them. Diamond shapes are used for representing conditional decisions. Last but not least, the standard also defines split and join gateways that create and, respectively, merge independent, concurrently executing workflow branches and depicts them as thick bars.

### 3.2.3. Event-driven Process Chains (EPC)

The event-driven process chains (EPC) [115] were developed by Wilhelm-August Scheer as part of the Architecture of Integrated Information Systems (ARIS) [182] framework. In comparison to the previous two specifications for designing process models, EPCs concentrate on the generic description of the execution domain of a process instead of its formal runtime semantics and implementation. They define the involved parties, the events that might happen and their respective outcome using a simple graphical notation and are thus much easier to comprehend by nontechnically minded professionals. EPCs are modeled using a specialized markup language called EPML which is widely adopted as part of the software systems developed by the SAP corporation.

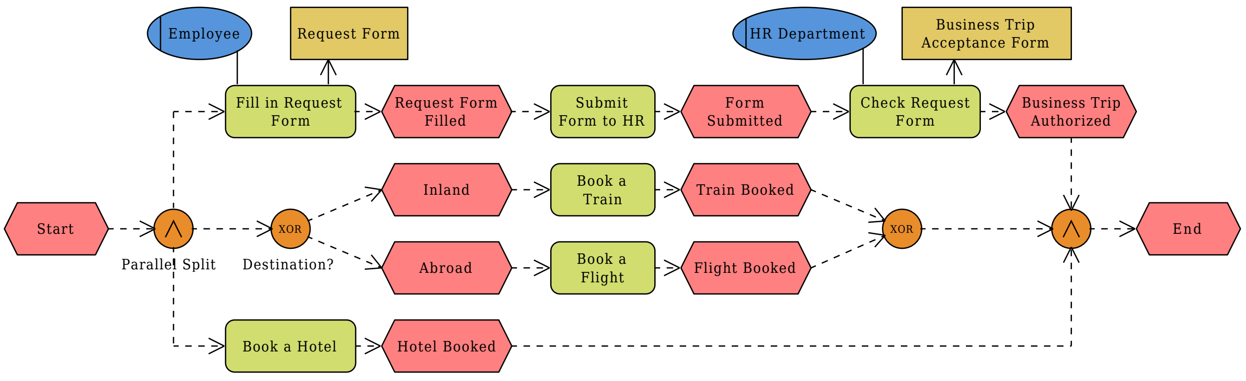


Figure 3.4.: A EPC model of a business process model for organizing a business trip

Every event-driven process chain is presented as an ordered graph of events, functions and conditional connectors and control flow arrows. Process events are depicted using hexagons with the names of the events inside them. They are passive components that define the current state of the process and have no influence on the flow of the work. Just like UML activity diagrams, process activities are designed using functions that are displayed using rounded rectangles and can additionally reference other EPC diagrams. Workflow gateways (splits and joins) are called conditional connectors and are graphically represented using a circle with distinct symbol referring to the underlying logical operation. For example, the parallel split gateway in Figure 3.4 creates three workflow branches that can be executed concurrently. The gateway tagged *Destination?* depicts an exclusive selection of only one branch at a time. In this case, the choice is based on the destination of the business trip. Another component of EPC models is shown as an ellipse with a vertical line and represents the responsible organizational units like the employee and the HR department. The last graphical component specified by the event-driven process chains is an information/resource object. It describes the result of a function and is drawn as a rectangle.

### 3.2.4. Business Process Model and Notation (BPMN)

Business Process Model and Notation (BPMN) [224, 7] is a process modeling standard that is constantly gaining popularity among the community. The standard was initially developed in 2004 by the Business Process Management Initiative (BPMI) and is nowadays maintained and extended to version 2.0 by the Object Management Group (OMG) standardization consortium. It supports an extended set of graphical notations, provides its own portable exchange format and defines a language for process execution. Its design goal is to combine the advantages of previous process modeling languages and specifications, i.e. petri nets, UML activity diagrams and EPCs, in order to enhance the modeling and automation of processes in many different fields of business, science and technology. In order to better illustrate the advantages of the standard in designing complex workflows, an extended version of the already presented process for requesting a business trip is shown in Figure 3.5. It features a larger set of the modeling components supported by BPMN including flow objects, swim lanes and artifacts.

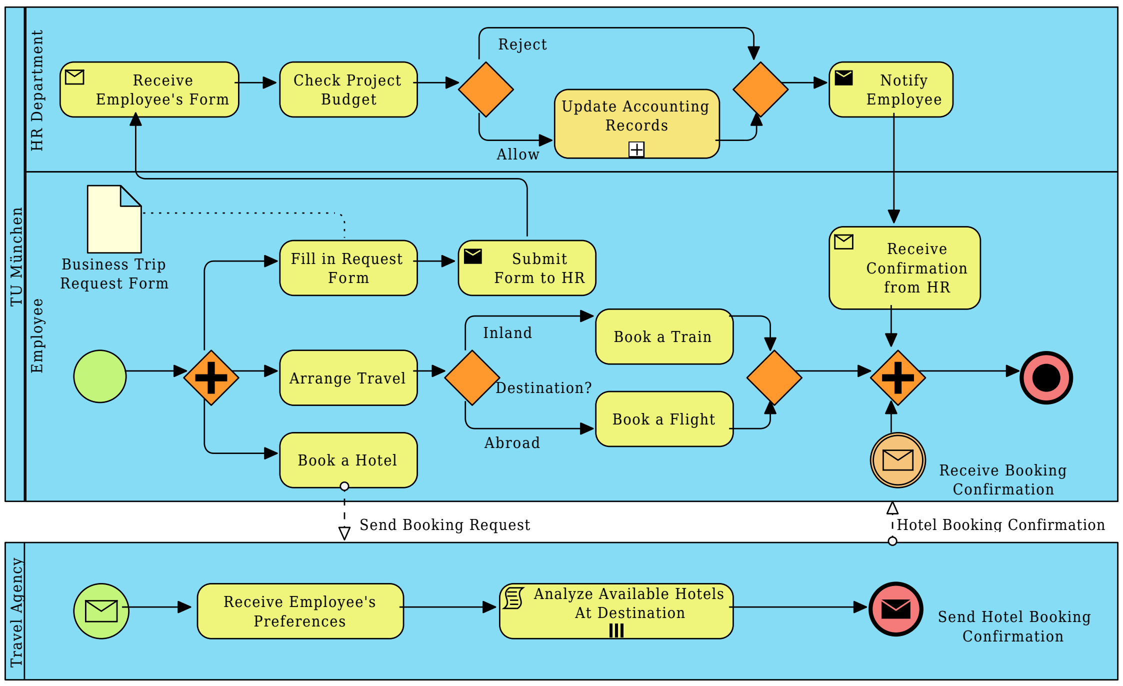


Figure 3.5.: A BPMN2 model of a business process model for organizing a business trip

One of the very basic types of BPMN modeling elements are the flow objects. These represent the active components in a process model and are internally subdivided into events, activities, gateways and sequence flows. They all have graphical representations and can possess additional attributes to help describe their implementation and execution requirements. As already mentioned in Chapter 3.1, events are depicted as circles. Depending on their position in the process models, they can be divided in three separate groups: start, intermediate and termination events. Each set can be further split in manual (e.g. the process state and stop events), triggered using external messages (e.g. the receipt of a booking confirmation from a travel agency), timer events, runtime exceptions, etc. (see Chapter 8.3 for additional details).

Activities are another type of flow objects supported by BPMN. They are graphically shown as rounded rectangles with the name of the task inside them. They can be simple autonomous tasks like the *Book a Train* activity or can refer to other process models like the *Update Accounting Records* task in the presented workflow. Moreover, the concurrent application of a task on multiple data entries or the definition of loops can be depicted using a single *MultiInstance* modeling element like the *Analyze Available Hotels At Destination* step of the given process model.

The last set of modeling elements in the flow objects group are the gateways and the sequence flows. The first are depicted using diamond shapes and can represent logical conditions or spawning/joining of separate work branches in the process. The second ones are drawn using directed arrows connecting other flow objects and describe the control flow logic of the process model.

The second major type of design components supported by BPMN are the swim lanes. They are adopted from the UML standard and are used to represent organizational entities. Swim lanes can be separated in two types: pools and lanes. A pool defines a separate organization like *TU München* and the *Travel Agency* in the model from Figure 3.5. Each pool can contain multiple lanes where each one describes a separate participant from pool's organization, e.g., the HR department or the employee requesting permission to go on a business trip. Characteristic for swim lanes is that the communication between two separate pools has to be modeled using messages (shown using dashed arrows in the figure) and not sequence flows. However, for designing the interaction between lanes in a pool, the usage of standard sequence flows is supported by the standard.

Last but not least, the BPMN standard supports the usage of artifacts for providing process documentation. Artifacts are always bound to a specific element but they have no direct impact on the design of the model or its execution. For example, the *Business Trip Request Form* artifact in Figure 3.5 does neither change the flow of the entire process nor influence the execution of the *Fill in Request Form* activity. It mainly serves as a documentation by providing a link to the actual request form. Another example of a BPMN artifact are the textual annotation of the sequence flow following a conditional gateway. The decision, which branch should be used, is taken by the *Destination?* gateway itself and the *Inland* and *Abroad* annotations simply improve the readability of the business process model.

### 3.2.5. Other XML-based Languages for Process Modeling

The Web Services Business Process Execution Language (WS-BPEL) [110] is a very popular specification for modeling the dynamic behavior of business processes. It was officially released in 2002 as a Business Process Execution Language for Web Services (BPEL4WS) [9] by the OASIS (Organization for the Advancement of Structured Information Standards) web services standardization consortium. It provides a portable XML-based format for specifying the autonomous interactions between web services. However, originally the language did not include support for manual tasks that are typically used in business processes and this limited the broad applicability of the language. As a consequence, this functionality was later on introduced as an extension called BPEL4People [117]. This extension is based on the WS-HumanTask [4] specification and its primary goal is to integrate different means of human interactions within the business processes and thus provide support for creating and executing role-based and long-running human activities.

XML Process Definition Language (XPDL) [143] is another XML-based language for the definition of business process models. It is developed by the Workflow Management Coalition (WfMC) and aims to complement BPMN in order to provide a standardized format for the exchange of process definitions between different business process management suites. In contrast to BPEL, this language not only supports the specification of the runtime environment, the participating entities and their execution constrains, but also contains information about the graphical layout of the BPMN components forming the process model. This results in a compact and very portable exchange format that unites the advantages of many other languages for process design and simulation. Therefore, it is implemented by the majority of today's BPMS in addition to the well-known BPMN 2.0 standard described in Chapter 3.2.4.

## 3.3. Business Process Management Suites

There is a big diversity of software systems for designing and automating business processes each with its own advantages and disadvantages. However, a trend of unification and standardization can be observed in the recent years. Many of the smaller BPM companies were acquired by big players like IBM and Oracle. This is a sign that the field is becoming more mature and established rather that just yet another technological hype. In the following pages, a short overview of two of the leading [1] commercial BPM suites will be given. Additionally, the features of the most popular, open-source competitor in the area of BPMS will be discussed.

### 3.3.1. IBM Business Process Manager

The Business Process Manager [93, 92] from IBM is regarded as one of the most scalable and complete software for designing, executing and optimizing business processes. After a successful acquisition of Lombardi Software by the software giant, the current state of the Business Process Manager is a combination of Lombardi's Teamworks, a software system previously regarded as being one of the best commercial workflow automation suites, and the prior BPM solution from IBM. It is available in three different editions, i.e. Express, Standard and Advanced, which cover

[1]The tools raking is based on the Magic Quadrant for Intelligent Business Process Management Suites [188] by the Gartner technology research and advisory company.

the needs from small companies with a couple of processes to multinational corporations with thousand of real-time business processes.

The BPM suite implements a central process center for designing, executing, monitoring of business processes and provides a web-based, collaborative interface. It has extended support of the BPMN and the BPEL standards and a centralized repository for process models. Using a performance data warehouse, knowledge workers have access to real-time monitoring scoreboards and process analytics. This way, they can detect process bottlenecks or identify new trends in order to optimize the resources that are critical for achieving company's goals. Moreover, the Business Process Manager can take advantage of other established IBM tools like the InfoSphere Master Data Management [95] and the Tivoli service management framework [94] and can interface with existing SAP [181] systems. However, the level of integration between the tools is not yet complete and still requires a lot of manual work. As a result, higher maintenance costs can be implicitly generated in comparison to the other commercial BPM suite reviewed in this chapter.

### 3.3.2. Appian BPM

Appian BPM [13, 12] is regarded as being one of the most powerful and yet user-friendly business processes management suites. In comparison to others, Appian does not concentrate only on the development of on-premises software automation systems, but combines the usage of cloud and mobile computing platforms in order to create a highly configurable and interconnected process management environment. Just like the IBM's BPM, Appian includes full support for BPMN and provides both a central repository for process models and a web-based management portal. Additionally, it supports powerful, in-memory analytics that can visualize the state of all processes in real-time, identify bottlenecks and even precisely predict the outcome of business activities based on data about their previous executions.

Furthermore, Appian BPM provides integration of live data streams from different social networks. This allows the creation of intelligent business process models that automatically monitor and react to information that people share on those networks. Other information sources like emails, messaging services or mobile applications are also supported.

### 3.3.3. JBoss Drools and jBPM

Drools [30] is a open-source, Java-based BPMS that is actively developed by JBoss and RedHat Inc. The project started in 2001 as a system for designing and executing business rules and was later on extended to include a flexible workflow automation engine with a centralized activity monitoring support, a complex event processing (CEP) [133, 134] module and an automated process planning and optimization component.

A central part in the overall system plays Drools Expert [197]. It provides a design environment for business rules and includes an internal rules execution engine. The rules can be defined using Java code or a declarative syntax based on the MVEL [29] expression language and are organized in logical groups called knowledge bases. Rules can range from very simple, e.g. where a single value is checked against a threshold, to highly complex data analyses as those needed by credit rating companies. In the early development, Drools Expert was using a brute-force approach for choosing the rules to be executed based on their preconditions. However, this approach proved to be highly inefficient and was replaced with the more advanced, pattern-based RETE [61] algorithm.

In the Drools platform, all process models and business rules can be stand-alone packages or they can be centrally managed using the Guvnor [199] component. The objects stored in this central repository can be easily accessed from other processes and design environments. Additionally, fine-grained access control is available so that even complex security requirements can be met. Another feature of Guvnor deals with versioning and comparison of stored knowledge bases. This aims at enhancing the tracking of the evolution of the designed processes throughout their complete life-cycle.

A major component of Drools is its business process management component. It is based on the jBPM 5 [200, 180] BPMS from JBoss and can be used either as part of the platform or separately. It has extensive support of the Business Process Model and Notation 2.0 standard and provides both web-based and integrated in Eclipse process editors. The workflow engine can be run stand-alone as a service or it can be embedded in an application. Additionally, it supports persistent process execution where the state of every activity of a workflow is stored. Similarly to the WS-BPEL specification discussed in Chapter 3.2.5, jBPM can integrate human participants in its process models using the WS-HumanTask specification. Due to its flexibility and open architecture, this business process management suite is also used within the PAThWay performance engineering automation tool as discussed in Chapter 8.

Quite often the traditional style for process design is inefficient in today's connected world. Diverse data streams can simultaneously deliver information that is relevant for the internal workings of processes and systems should be able to react in close to real-time. Moreover, the order in which information arrives from the given data streams can be of a major importance to the successful execution of a process. For example, a fraud detection system concurrently monitors the bank transactions of thousands of customers but should only tag a small subset of them depending on, for example, their causality and timings. In order to cover all these requirements, the Drools platform features a CEP component called Fusion [198]. It can asynchronously process data from multiple streams and identify temporal and hierarchical relationships between them allowing the design of dynamically adapting process models. Like most of the other components, Drools Fusion can also be used separately or within business process models.

Last but not least, the Drools platform includes a powerful constraint satisfaction [207] component called OptaPlanner [202]. It provides multiple optimization algorithms for the efficient planning of resources such as process execution schedules. Due to its generic design, it can also be used to solve many other optimization problems, e.g. route planning, bin packaging and portfolio optimization, using different maximization or minimization techniques. Similarly to the Expert component, the analyzed models together with their constrains can be also designed using either Java code or Drool's business rule definitions.

## 3.4. Scientific Workflow Automation Tools

The second major type of automation systems deals with the design and execution of scientific processes. As discussed by Ludäsche et al. in [136], the majority of scientists are interested in the acquisition, distributed processing and management of high volumes of data commonly using Grid-based environments rather than automating activities and complying with regulations which are the main design goals of business process management suites. As a result, a separate research branch was created with the aim to investigate and design software systems for modeling dataflows and their transformations. In the course of time, many such systems were developed and adopted by the scientific community [16]. Even though some provide proper integration with user's development environments such as Palladio Workflow Engine [174], Modeling Workflow Engine (MWE) [80] and PTP's External Tools Framework (ETFw) [137], the majority still have a steep learning curve and depend on their own workflow modeling specifications instead of following the already established BPM standards. To give a broader overview of the current process automation field, the three most popular scientific workflow management systems will be shortly presented in the following sections.

### 3.4.1. Kepler

Kepler [8, 135] is an open-source, Java-based workflow management system for the automation of scientific simulations. Its development started in 2002 within the National Center for Ecological Analysis and Synthesis (NCEAS) with the goal to create a domain-specific modeling and execution language that is as close to the application developers as possible. Due to the strong data dependence of a typical scientific workflow, the Kepler system was not implemented around any of the existing at that time business process management suites but instead extended the Ptolemy II [53] dataflow modeling system.

The system provides a visual editor for modeling the execution of scientific processes using a specialized, XML-based specification called the Modeling Markup Language (MoML) [128]. Being an extension of Prolemy II, Kepler provides a extendable architecture based on plug-ins. Each workflow step is presented as an *Actor* and all actors are controlled from one central *Director*. The actors have input and output connectors and are used to represent the logic behind scientific workflows such as data analyses and transformations, database access routines and custom code snippets. The director is used to represent the interconnection between the actors and their execution constrains.

Additionally, the Kepler system is designed to support distributed processing and management of data. It can interface with web services using the Web Services Description Language (WSDL) [42] standard and Grid-based computational environments. Remote execution is controlled using the Globus Grid Toolkit (GT) [62] and the Storage Resource Broker (SRB) [18].

### 3.4.2. Taverna

Taverna [151, 208, 90] is another open-source workflow management system that targets explicitly the needs of the scientific community. The first version of the tool was released in 2004 by the myGrid e-Science research group from the United Kingdom. Similarly to the Kepler system, the design goal of Taverna is the modeling and execution of data-centric scientific workflows. However, it supports only web services using the WSDL standard and no Grid-based execution environments. The steps defining a particular workflow are also called *processors* instead of actors. All parameters and input/output connectors of processors are designed using the second version of the Simple Conceptual Unified Flow Language (SCUFL2) [150]. This specification provides control and dataflow links for the definition of execution constrains between processors.

Taverna supports two execution modes: a stand-alone and shared ones. The first one is supported using the Taverna Workbench which is the cross-platform, Java-based user interface of the system. It provides an execution engine and a graphical designer for scientific workflows. The workbench also includes an extensive repository of both local and public web service definitions which can be easily integrated in workflow models.

The Taverna Server is another method for interfacing with the system. It provides a web-based, multi-user execution portal where scientists can upload and remotely execute their SCUFL2-based workflow models. The server will then run the processes on behalf of the users and will make the results available on the web portal upon successful workflow completion.

### 3.4.3. Pegasus

Another very popular and actively developed workflow management system is Pegasus (Planning for Execution in Grids) [54, 55]. Its main goal is the automatic mapping of high-level scientific workflows to distributed computational resources like Grids and Cloud computing services and their consecutive execution. It uses an XML-based modeling language called DAX for designing process workflows. These DAX models can be created either manually by advanced users or automatically generated from a high-level definition of the required data, their transformations and expected results. The later approach is based on the Chimera [63] system and its Virtual Data Language (VDL) specification.

The main different between Pegasus and the previous two workflow management systems is the execution engine. Instead of requiring the precise definition of all required resources and their locations, the Pegasus system uses centralized location services like the Globus Replica Location Service (RLS) [41] to identify and access distributed data. Furthermore, it abstracts the technical details of supported execution middleware systems, i.e. Globus, Condor or Amazon EC2, by autonomously interfacing with them on behalf of the user. This way, the scientists can concentrate on modeling their experiment workflows and do not have to learn yet another technology.

### 3.4.4. Domain-Specific Automation Languages

Process modeling and automation is not only important in the fields of business administration and computer science. Many other scientific areas like computational biology, chemistry and economics depend on the execution of multiple numerical experiments to gain deep insight into their research areas and are also very suitable candidates for automation. As a result, new domain-specific process modeling languages started appearing that are specifically optimized for the needs of those research groups.

The Simulation Experiment Description Markup Language (SED-ML) [123, 218] is such an example from the area of computational biology. It implements the standard for Minimum Information About a Simulation Experiment (MIASE) [217] and aims at formalizing the design, the execution and the documentation of simulation workflows in order to support scientists in creating reproducible research results. SED-ML uses an XML-based specification and provides the following modeling units:

- Computational Models - a specification of the required numerical models, the location of their definitions and their parameters. Models can be designed using different domain-specific languages such as SBML [89], CellML [149] and MathML [14].

- Simulation Algorithms - a definition of the applied numerical algorithms together with their runtime configuration.
- Tasks - a specification of the experiment activities, i.e. workflow steps, and their internal relations such as computational models and simulation algorithms.
- Data Generators - a configuration for the required post-processing operations on the results of a task element like data normalization.
- Output Elements - a definition of data aggregation steps and destination formats for storing the workflow results.

Even though SED-ML was initially developed for the needs of the computational biology community, it can also be applied to other scientific research areas due to its generic modeling support. In the beginning, the language started as a plain, textual specification for augmenting research publications with precise experiment information. However, different software tools and libraries were developed through the time for the graphical design, validation and autonomous execution of such simulation workflows.

Another domain-specific workflow language is the OMF Experiment Description Language (OEDL) [171]. It targets the needs of communication technology research groups and is used to design and execute reproducible test experiments for the analysis of new networking technology.

LINDA [138] is another example for a experiments definition language from the field of pulse fusion physics. It allows researchers to define the required hardware and the complete data acquisition and analysis workflow behind an experiment. The models have a plain text format and are compiled at runtime to execution objects by an execution engine called MDS-Plus [60].

Machine learning (ML) techniques are the major research tool in the field of computational intelligence. These methods require large amounts of data and depend of multiple independent experiments. Consequently, another domain-specific language named ExpML [26] was developed by the community in order to structure the way data mining experiments are designed and conducted. It uses an XML format and, similarly to SED-ML, consists of different modeling elements that can be used to describe the whole life-cycle of ML studies. Recently, the ExpML language became part of a new, web-based platform from KU Leuven called OpenML [130]. Its main goal is to create a central database for machine learning research in order to ease the global access to data mining information, create a collaborative environment and thus foster further development of the field.

Chapter 4

# Supportive Software Development Tools

**Contents**

**4.1. Revision Control Systems . . . . . 37**

**4.2. Client-Server Repository Model . . . . . 38**

4.2.1. Concurrent Versions System (CVS) . . . . . 39

4.2.2. Subversion . . . . . 40

4.2.3. Perforce . . . . . 40

**4.3. Distributed Repository Model . . . . . 41**

4.3.1. BitKeeper . . . . . 42

4.3.2. Git . . . . . 43

4.3.3. Mercurial . . . . . 43

## 4.1. Revision Control Systems

Optimizing parallel applications can be quite a tedious work and normally requires plenty of source code transformations until a desired result is achieved. Sometimes improving one part can lead to instability or decreased performance in other parts of the application. This will consecutively require that all new changes are rolled back to stabilize the software.

Moreover, the diversity of parallel architectures often requires the existence of different versions of the very same code. In order to support the application developers in their daily work and help them deal with this versioning complexity, version control systems (VCS) were created. They form a very important group of tools for the software developer and can be additionally used to provide a clear track of the tuning process.

**Definition 4.1.** A *version control system* (also commonly called revision control system) is a software storage system that tracks changes to files and folders store in it and provides easy access to any past version of the stored entities. These systems can be stand-alone such as CVS, SVN and GIT or they can be embedded in other software tools. The two major types of version control systems are centralized and distributed.

In the following subsections, the two major types of VCS will be compared. Tools from both groups will be analyzed in more detail in order to evaluate their usability in tracking the tuning progress of parallel applications.

## 4.2. Client-Server Repository Model

The client-server repository model is the typical approach to managing the evolution of software systems by tracking incremental source code modifications. This method is based on the usage of a centralized server that stores the current version of a software project together with its development history.

**Definition 4.2.** A software *repository* is the location where software artifacts such as source code, resources and data are stored. These contents made available to a broader public to facilitate easy software distribution and are commonly versioned using version control systems.

In order to start working, developers have to check-out a complete copy of the source code, called a working copy, directly from a central server by connecting over a network to it. As seen in Figure 4.1, the developers have to check-in or commit their modifications directly to the same server, thus allowing a centralized control over the current state of the project. Moreover, the server normally accepts only commits to the latest version of the source code and so requiring users to regularly update their local working copies with the latest modifications. During an update, the newest changes are often automatically merged with the local files by the revision control system. However, sometimes a conflict can arise which will require manual integration of the changes. This usually happens when the developers do not keep their working copies up-to-date or in cases of many complex code variations.

Upon a successful commit, the revision control system automatically increments the version numbers of all involved files and stores a user-supplied description of the checked-in code together with author's name and time. Moreover, the server can be configured to execute special commit hooks before and/or after a check-in. These hooks are normally external scripts for manipulating the new source code and can be used to assure specific coding style, trigger email notifications, run regression tests, etc.

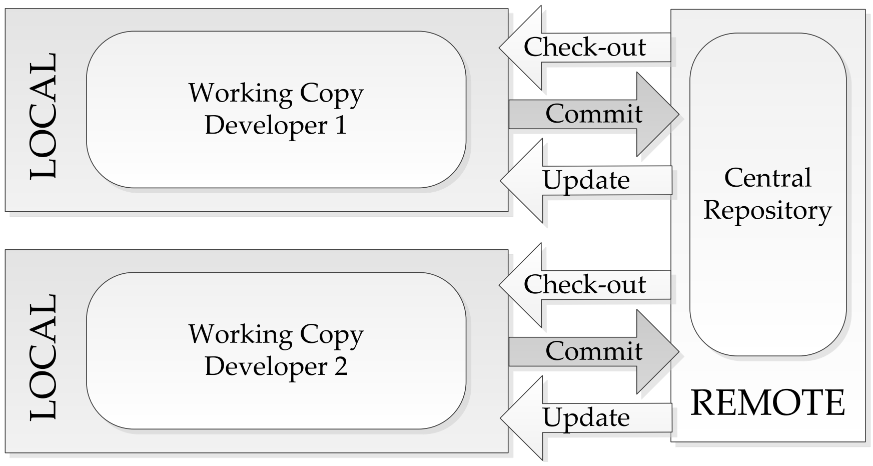


Figure 4.1.: Client-Server Version Control Workflow

### 4.2.1. Concurrent Versions System (CVS)

The Concurrent Versions System (CVS) [82] used to be the trend setter in the early ages of revision control systems. It was initially developed by Dick Grune as a collection of shell scripts in July 1986. Due to its open-source design and support for commits over the network, it became the main VCS for many distributed open-source projects.

CVS implements all features described in the previous section and uses a central server for the source code management. A working copy in CVS is represented as an ordinary directory tree on the local system. The system also creates and maintains extra files in order to help it carry out commits and updates and stores these in a separate CVS folder at each level of the tree of the working copy.

In the course of time, CVS proved to be instable for many of today's development methodologies. It does not support atomic commits and each check-in containing multiple files is in reality a collection of consecutive commits - one for each file. This creates a bigger overhead to assure consistency of the repository and often requires special administrative workarounds to handle even simple tasks. Moreover, due to the fact that CVS assumes the majority of work takes place on one main branch, its branch and merge operations are very expensive and further increases the management overhead.

These combined with the lack of support for easy code refactoring by ,e.g., moving or renaming files and directories makes the Concurrent Versions System generally not recommended for new software projects. In many organizations it was already replaced by its more modern successor, i.e. the Subversion system, which is discussed in the following subsection.

### 4.2.2. Subversion

The Subversion (SVN) [168] project was founded in 2000 by CollabNet Inc. and it was, later on in 2009, moved to the Apache Software Foundation. It is the most popular open-source version control system nowadays. It aims at replacing the old CVS system by providing similar functionality but with improved operability.

The Apache Subversion system is written in ANSI C and consists of a set of libraries. It uses APR, the Apache Portable Runtime library, as a portability layer and as such will run anywhere APR runs, i.e. all modern flavors of Unix, Linux, Windows and MacOS X.

Just like CVS, the Subversion system represents its working copies as a collection of user and system files and folders. In the past, each directory in the working copy contained a hidden .svn subdirectory that stored the repository metadata. However, this structure was recently changed and similarly to other modern version control systems SVN now uses only a single subfolder for all its meta-information.

Repositories can be accessed directly when they are stored on the local file system or through the network. In the second case, Subversion provides many different access protocols with varying level of security. It can use a custom protocol to connect to a standalone svnserve server and can additionally tunnel the connection through Secure Shell (SSH) network protocol. As a result, it can provide a very secure access to remote repositories.

However, in contrast to a simple branching operation, a merge operation in Subversion can be quite complicated and lead to unexpected results. Even though there are many recent improvements with this respect, it is still often not possible to do such operations automatically without a great deal manual inspection and modification of the affected source code. This limits its applicability in highly distributed and dynamic projects where multiple branches are required in order to allow parallel software development.

### 4.2.3. Perforce

Another revision control system using the client-server model is Perforce [226] SCM toolkit. Even though it is a commercial revision management software developed by Perforce Software Inc., it also comes with a free license for small software projects and academia. Due to its robustness and quality, it is the tool of choice for many commercial companies world-wide including Boeing, IBM, SAP, Google, etc.

Among its strengths are its powerful branching/merging functionality, the rich palette of version history representation tools, and the build-in bug tracking system. It also provides native application programming interfaces (API) for C++ and Java and language extensions for Perl, Python, Ruby, etc. Furthermore, the Perforce system can interoperate with the GIT distributed version control system presented in Section 4.3.2.

Last but not least, it can integrate with other tools like Eclipse, Adobe Photoshop, and Autodesk's 3D Studio Max in order to come closer to the working environment of the users. It runs on a wide range of platforms, including Windows and almost every variation of Linux.

## 4.3. Distributed Repository Model

Since several years, a trend to move away from the centralized client-server model to a distributed one [159]. In contrast to the old approach, distributed version control systems (DVCS) use a peer-to-peer architecture. Rather than a single, central repository on which developers synchronize their work, each one checks out their own full copy of the repository. This local version includes not only the main version of the code but also all available branches and their revision histories. One big advantage of this data redundancy over the traditional revision control systems is that all copies of the repository also function as backups or mirrors of the source code and the metadata. This implicitly leads to higher availability and reliability of the repository.

As opposed to the client-server approach, a big advantage of DVCS is that that the majority of commands operate locally and are therefore much faster than the client-server model. A network connection is only required when publishing to or fetching changes from other repositories (see Figure 4.2).

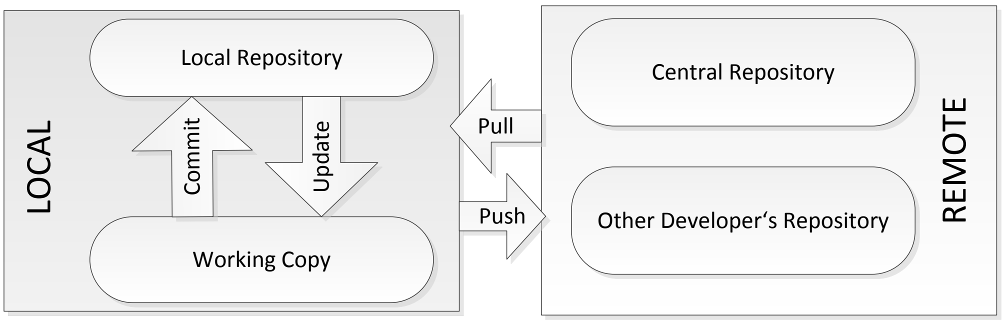


Figure 4.2.: Distributed Version Control Workflow

A DVCS is useful in very large software projects because of its enhanced management support for supports rapid branching and merging. This not only makes the delegation of programming tasks easier, but also allows the parallel non-linear development of different features and their selective inclusion in the main product line. Moreover, since users commit only to their local repositories, the do not need to constantly synchronize with the work of others and can freely experiment different ideas without interfering with the rest.

Depending of requirements of a project, changes can then be shared using the standard approach of syncing to the central repository or can be propagated to other developers by exchanging patches with them. This allows a more collaborative environment in which external parties can contribute to a project without having direct access to the development repository. As a result, not only the administration overhead is decreased but also a better separation of control and thus a increased security is achieved.

Last but not least, distributed version control systems have enhanced support for code refactoring. This is done by tracking global content changes instead of just individual files. This makes it much easier to move or rename the files, to extract source code regions into separate files or to track a function moved from one file to another.

One disadvantage of distributed version control systems is that they do not support central locking of separate resources which can lead to merge conflicts when dealing with non-mergable files in, e.g., binary format. Another drawback of DVCS is the initial check-out of a repository. This process can take much longer in comparison to client-server model due to the bigger amount of metadata that has to be copied locally. However, since this is often a one time task, it has a very little impact on the overall development process.

### 4.3.1. BitKeeper

BitKeeper is developed by BitMover Inc. and it is regarded as the tool that laid the foundations of all modern distributed version control systems. Due to its scalable architecture and enhanced support for distributed development teams, the tool was quickly adopted by many Free and Open Source Software (FLOSS) projects. It introduced for the time features like atomic commits of multiple changes and the ability of using local repositories that can be synchronized with the central server at a later time.

The producer of BitKeeper initially offered it free of cost to many open-source projects among which the most famous one is the Linux kernel. However, this free license was very restrictive in that it not only forbid its users to work on projects connected with the development of other VCS, but also provided limited support for comparison among revisions. As a result, BitKeeper became probably the most disputed distributed revision control system at that time and it faced broad resistance from the FLOSS community. This opposition was then further amplified when the company decided to ultimately revoke the free license for all FLOSS projects in 2005. Consequently, many Linux developers decided to abandon BitKeeper and this eventually lead to the creation of the two most popular open-source revision systems nowadays - Git and Mercurial.

### 4.3.2. Git

Git [37] is currently one of the mostly discussed and frequently used distributed version control systems. It was initially developed by Linus Torvalds in 2005 to replace the BitKeeper system that was used for tracking the development of the linux kernel. Its main design goals are speed, efficiency and usability on very large distributed projects with thousands of developers.

Like most other modern distributed version control systems, each local working copy of a Git repository contains a complete track of the code evolution together with meta information about authors, date and time of the changes, etc. This historical information in stored in a hidden top-level folder and can be either synchronized directly between separate developers in the form of patches or using a central server. In order to insure data consistency and guarantee that no previous versions of the code were tempered between commits, Git provides a build-in cryptographic authentication as well.

Remote repositories can be easily accessed via a specialized Git protocol and the connection can be additionally secured using SSH. Another option to publish a repository to the public is to use the Hypertext Transfer Protocol (HTTP) or the File Transfer Protocol (FTP) protocols.

One of the fundamental design goals of the toolkit is speed and efficiency when working with large projects. In comparison to the Concurrent Versions System, Git supports atomic commits where all changes are stored at once and not split into multiple commits on a per-file basis. Additionally, it uses a highly scalable, compressed format for its internal metadata which allows the efficient management of large projects with tens of millions of revisions.

Git provides native support for all major Linux and UNIX distributions. However, it has a rather limited functionality on Windows machines where one has the choice between the specialized plug-ins for Eclipse and VisualStudio or the recently released Git Extensions that aim to integrate the system with Windows Explorer.

### 4.3.3. Mercurial

The Mercurial [160] distributed version control system is regarded to be the main opponent of Git. They both have quite similar design and functionalities but differ in the implementation and the resulting architecture.

Just like Git, Mercurial development started after BitKeeper, the versioning system that was used at that time for the code of the Linux kernel, became paid in 2005. At that time BitMover Inc, the company developing BitKeeper, refused to give and even sell licenses to the Linux kernel developers making the further usage of the system impossible. This motivated Matt Mackall to develop a brand-new distributed version control system that will track the evolution of the Linux kernel. However, the kernel developers decided later on to adopt Linus Torvalds's Git instead of Mercurial. Nevertheless, its popularity is still growing as it is adopted in large projects like Mozilla, OpenJDK, NetBeans, and Python.

In contrast to Git, the Python programming language was chosen for the base implementation of the Mercurial DVCS. An effect of this design decision is the broad portability of Mercurial to all major operating systems and its easy integration in other tools. From the functional point of view, Mercurial has similar advantages like Git but a bit problematic merging of branches. It again uses one single top-level folder for storing historical information about the repository and applies cryptographic techniques to differentiate between revisions and assure code consistency.

Mercurial also supports the concept of Super projects [193]. It can reference local or remote Mercurial repositories and can additionally include external SVN repositories. This allows the configuration of "meta" repositories which will host and track the progress of many other projects, thus making it very useful in heterogeneous environments.

Chapter 5

# Related Work

**Contents**

**5.1. The Need for Parallel Programming . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46**

**5.2. Performance Engineering Automation . . . . . . . . . . . . . . . . . . . . . . . . . . 47**

**5.3. Performance Engineering Tools . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49**

5.3.1. Periscope . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49

5.3.2. Scalasca . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53

5.3.3. Vampir . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55

5.3.4. TAU and PerfDMF . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56

5.3.5. Score-P . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58

5.3.6. IBM HPCS and the BDE / SDE / SIE . . . . . . . . . . . . . . . . . . . . . 60

5.3.7. HPC Toolkit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

**5.4. Performance Tuning Libraries and Frameworks . . . . . . . . . . . . . . . . . . . . 62**

5.4.1. Automatic Performance Tuning HPC Libraries . . . . . . . . . . . . . . . . 62

5.4.2. Search-based Performance Tuning Tools . . . . . . . . . . . . . . . . . . . . 62

5.4.3. Automatic Performance Tuning Frameworks . . . . . . . . . . . . . . . . . 63

**5.5. Eclipse and the Parallel Tools Platform . . . . . . . . . . . . . . . . . . . . . . . . . 64**

5.5.1. Eclipse Integrated Development Environment . . . . . . . . . . . . . . . . 64

5.5.2. Parallel Tools Platform (PTP) . . . . . . . . . . . . . . . . . . . . . . . . . 65

## 5.1. The Need for Parallel Programming

For a good part of the history of computing, application developers have relied on the performance improvements achieved by a single processor due to its constantly growing clock frequency and instruction-level parallelism to deal with increasingly complex computational problems. The power dissipation of CPUs , however, is proportional to the square of the operating voltage and the frequency. Furthermore, as part of the normal operation every digital circuit, there is an intrinsic capacitance whose speed of charging and discharging depends directly on the applied voltage, i.e. higher voltage is required to achieve faster switching frequency. As a result, the type of processors based on frequency scaling design stopped being manufactured some years ago.

Consequently, around 8 years ago, a major shift in the design of modern processors happened - CPUs with multiple computing cores but much lower frequency started appearing on the market. This course continued and nowadays multi-core processors are everywhere ranging from mobile phones, over portable computers, up to high-end HPC systems. As a result, a totally new approach to software development became necessary as developers could not rely any longer on the underlying hardware to accelerate the execution of their codes but had to manually modify their algorithms to be able to take advantage of those additional processing units.

Naturally, different programming paradigms that used to be popular mainly in the field of high-performance computing started being adopted by the wider software development community. Standards like OpenMP (Open Multiprocessing) [51] and Message Passing Interface (MPI) [145], specialized libraries like Intel's Math Kernel Library (MKL) [106] and frameworks for distributed computing like Apache Hadoop [140] are becoming an integral part of today's software development toolkit. All these utilities enable the simultaneous usage of not only multiple processor cores but also of many interconnected machines. Some of them require little changes to the sequential version of an application and so are very suitable for the broad majority of developers. Others require considerable changes in the program's logic and data structures.

**Definition 5.1.** *Performance engineering* is a complex process which includes multiple performance modeling, prototyping, static and runtime program analyses and performance tuning tasks. It can be part of the perfective maintenance phase of the software development life-cycle of an application. It most commonly spans over whole SDLC.

However, all these parallel standards and libraries depend upon a deeper knowledge in order to achieve the best performance out of the underlying hardware and software architectures and thus be able to explore even bigger scientific problems in shorter time. Due to this inherent complexity, a whole research field in computer science is dedicated extensively on developing expertise about different parallel hardware and software systems, their dependencies and the effect they have on the performance of HPC applications. As a outcome of the extensive research done in the field, many powerful performance analysis and tuning tools were developed. These tools effectively combine the know-how of many HPC experts in order to support application developers during the optimization of their code faster and in an uncomplicated manner.

## 5.2. Performance Engineering Automation

Performance engineering is a complex and usually a very long running, iterative process. It plays a central role in the area of high-performance computing and cannot be easily ignored as commonly done in other areas of software development. Due to its direct impact on the effective usage of parallel systems and the resulting ability to run even more complex scientific simulations, a lot of research efforts are devoted to enhance the overall performance engineering process. In the course of time, many different tools were developed to support the users in every step of the process from the abstract modeling of the expected performance of highly parallel applications, through the dynamic analysis of their execution to the implementation of successful tuning strategies.

As it will be seen in the following sections, the existing performance engineering tools have become fairly complex. Nowadays, they successfully automate many activities in order to provide the users with a deeper but easier to comprehend insight about the execution behavior of analyzed applications. However, most of currently state-of-the-art performance analysis and tuning tools are very specialized in what they do and thus users must often apply many different ones in order to finally come up with the best performing version of their codes. Moreover, the majority of the existing tools and HPC systems require extensive training before they can be productively used. All these factors actively impact the successful optimization of parallel applications and so indirectly hinder the development of many compute-intensive scientific domains.

Consequently, this work tries to improve the current situation by combining some of the established business process modeling standards presented in Chapter 3 with different tools for performance engineering. Even though the idea of HPC process automation was previously investigated in the area of Grid computing (see Section 3.4), the concept of workflows for performance analysis and tuning is rather new without much of published research. One of the main contributions in this area comes from IBM and their Eclipse-based environment for parallel programming - the Parallel Tools Platform (PTP). This platform not only supports application developers in detecting common parallelization problems such as deadlocks and improper synchronization points, but also allows enhanced remote development which is very common in the area of HPC. As discussed in Section 5.5.2, is also features a workflow modeling component called the External Tools Framework (ETFw). However, similarly to the majority of workflow tools for Grid automation, it specifies its own modeling language which is rather restrictive and not reusable with other automation tools. Moreover, it does not currently provide any support for integrating existing tuning frameworks or any form of human interactions as part of the designed workflows.

Additionally, it has to be mentioned that some of the tools presented later in this chapter also include support for executing common performance engineering processes. For example, the Periscope performance analysis tool can autonomously run OpenMP-based scalability analysis and provide aggregated data about the resulting performance behavior (see Section 5.3.1). However, this feature is an internal part of the tool and as such it is not easily adaptable to other use cases or processes. Other systems like the Score-P monitoring infrastructure, which is presented in Section 5.3.5, specify high-level guidelines for approaching the complex problem of performance engineering but currently do not provide any means for the automatic execution of the involved steps.

When it comes specifically to creating a centralized environment for the management and the storage for multiple performance experiments, many different tools were developed in the course of time. Among the popular examples are PerfTrack [113, 118] from the Lawrence Livermore National Laboratory (LLNL), PerfDMF [88] developed by the Performance Research Lab of the University of Oregon, and SCALEA [206] which was created by the Distributed and Parallel Systems Group from the University of Innsbruck. In contrast to this work, however, the first two mainly target the storage and comparison of raw performance data and do not provide support for automatic execution and monitoring of performance experiments. At the same time, even though the last one is capable of autonomously instrumenting users code and running performance studies, its design is centered around the toolkit developed by the SCALEA research group and has very limited support for integrating other performance engineering tools. Furthermore, it has no support for modeling new, user-specific workflows and thus cannot be used for automating the previously discussed performance analysis and tuning cycle. Other similar examples include Zenturio [169], PPerfDB [46], and Prophesy [195].

The next sections will provide a deeper overview of the current state-of-the-art performance engineering tools, their features and key design decisions involved in their development in order to help the reader to better understand the field of performance analysis and tuning. After that, the Parallel Tools Platform will be shortly introduced as an example of how others aim to automate and integrate the execution such tools in order to enhance the overall performance engineering process.

## 5.3. Performance Engineering Tools

Performance engineering tools are the instruments required to successfully develop applications capable of efficiently utilizing the today's and future computing systems. They consist generally of two major components. First, it is necessary to be able to monitor a system and to capture the data that reflects its behavior. Second, it is necessary to process such data in order to extract meaningful information out of it. The first component has a strong dimension of infrastructure and data handling mechanisms and its major challenge is to achieve an overall good scalability. The second component is mostly related to data analysis: providing algorithms for identifying and understanding all the different factors that affect the runtime performance such as application's structure, machine characteristics and their interactions.

**Definition 5.2.** *Profiling* is a performance analysis approach in which raw performance data is aggregated over multiple dimensions such as time, processes, threads, code regions, etc. For example, the number of cache misses for the individual functions in each process. Additionally, some performance analysis tools provide time series of snapshots of profile data in order to allow the analysis of time-dependent variations of the performance behavior [75].

**Definition 5.3.** *Event Tracing* is a detailed performance analysis technique during which information about individual application events such as the start or the completion of a function or the transmission of a message are recorded. Each event is recorded in a trace file together with a precise time stamp and additional meta-information about it [75].

Based on the analysis methodologies applied by the different performance engineering tools, they can be subdivided into groups supporting only runtime profiling techniques, others providing detailed event tracing functionality and ones which combine both approaches. Moreover, some performance engineering tools and libraries go one step further and try to automatically optimize the runtime performance to the given execution environment. In the following subsections, an overview of the current state-of-the-art performance analysis and tuning tools will be given. Each section will present the design ideas behind the tools and how these aim to solve the previously mentioned challenges in order to provide an efficient performance engineering environment.

### 5.3.1. Periscope

Periscope [22] is a performance analysis tool developed at Technische Universität München. It is a representative for a class of performance analysis tools that automate the overall analysis procedure. Periscope is an online analysis tool and works in a distributed fashion. Currently, the toolkit can analyze single-node performance, MPI communication behavior, OpenMP threading, as well as the scalability of highly parallel C/C++ and FORTRAN applications. Furthermore, an automatic tuning framework for highly parallel code is being developed based on the Periscope tool (see Chapter 5.4.2).

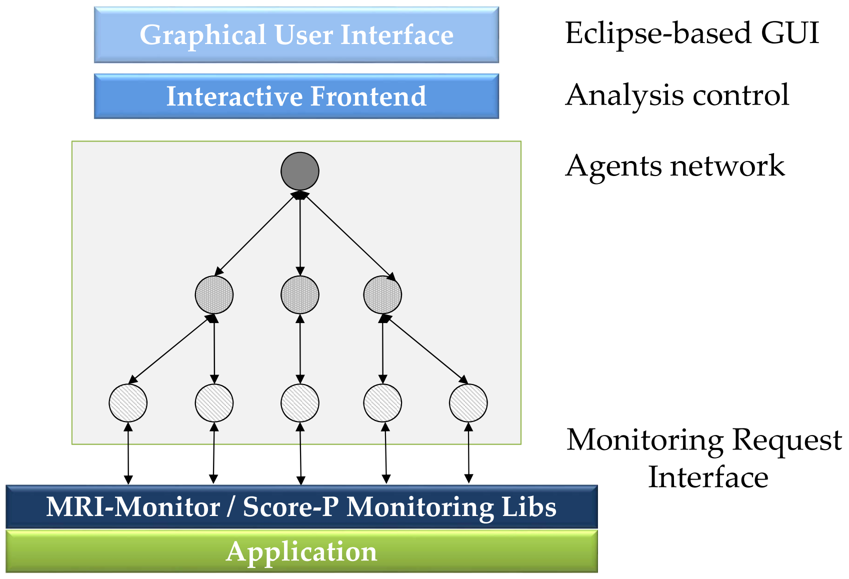


Figure 5.1.: Distributed hierarchical architecture of Periscope

As it can be seen in Figure 5.1, Periscope's architecture can be separated in multiple interdependent software layers. The topmost layer provides the performance engineer with a convenient graphical user interface for the analysis of collected performance data. The second tier in the architecture is presented by a front-end control interface which serves as the main access point to the tool's measurement framework. Using this interface, the user can easily configure, execute and monitor different performance engineering studies. Additionally, this component is responsible for building a platform-optimized hierarchy of distributed analysis agents [77]. Represented as the third layer in the figure, this tree structure creates an efficient processing environment in order to allow the tool to monitor the performance behavior in real time while the analyzed code is being executed. Characteristic for Periscope is that each node in the hierarchy has a different behavior and responsibility. The leaves, for example, are responsible for collecting low-level performance metrics such as CPU hardware counters, number and size of MPI messages and others. Furthermore, they process and evaluate the gathered data against a collection of predefined performance problem descriptions and so generate a set of high-level performance properties. After that, these properties are propagated upwards in the tree structure where they get clustered using specialized algorithms. As a result only a high-level abstraction of application's performance is finally reported to the user, thus considerably reducing the data load to be examined by the user.

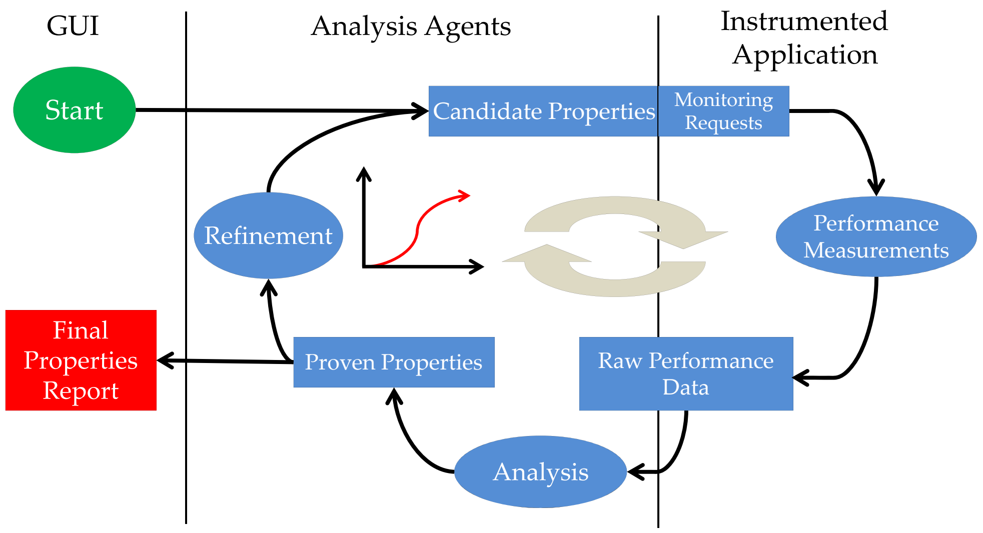
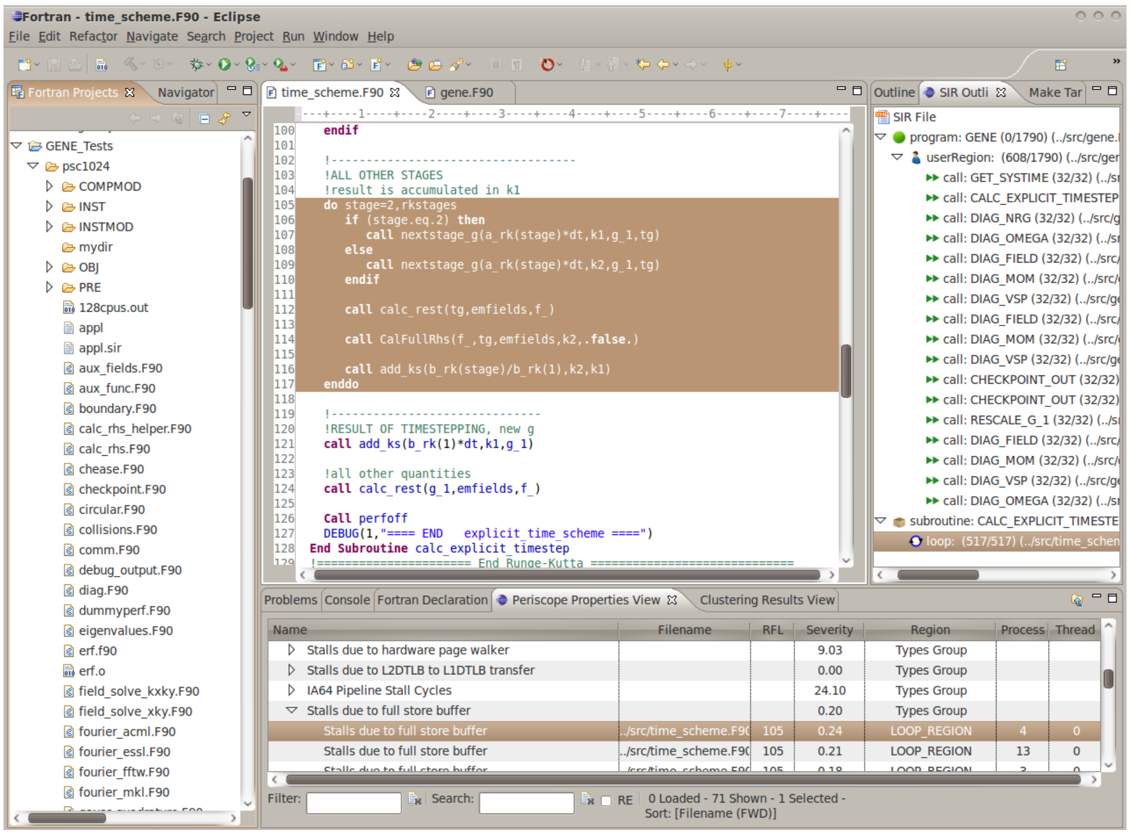


Figure 5.2.: Iterative online performance analysis using Periscope

Periscope uses high-level search strategies [76] to identify existing performance bottlenecks in the runtime behavior of applications. They consist of a set of instructions for the analysis agents and prescribe how the monitoring should be done, i.e. code sections should be analyzed and what kind of performance metrics should be collected from them. Some strategies perform a single-shot analysis where the application or the selected code region is executed only once. However, others require multiple runs in order to gather all required data (see Figure 5.2). For example, during the multistep analysis strategy of CPU stall cycles, the tool can automatically exploit the logical hierarchy of processor's hardware counters in order to further refine the discovered properties and thus provide an in-depth information about the runtime behavior. A very important requirement for these iterative strategies is, however, the existence of a repetitive program region such as the time loop in scientific simulations where each iteration calculates one time step of the given problem.

Additionally, the Periscope performance analysis tool provides an Eclipse-based graphical user interface (GUI) [166]. It provides an integrated, close-to-the-user environment for exploring collected performance information and simultaneously links the detected bottlenecks directly to their location in the original source code. As it can be seen in Figure 5.3, the GUI of Periscope provides two jointly working views that not only display the identifies problems, but also show the structure of the instrumented application so that the performance engineer can easily navigate within the code. Instead of providing multiple different charts like other performance analysis tools, Periscope uses a highly-configurable table for the presentation of detected performance properties. This table provides the user with different options to restructure and customize the displayed performance data so that maximum knowledge can be deduced out it. Among those features, the most commonly used are:



Fortran - time_scheme.F90 - Eclipse
File Edit Refactor Navigate Search Project Run Window Help
Fortran Projects
Navigator
GENE_Tests
psc1024
COMPMOD
INST
INSTMOD
mydir
OBJ
PRE
128cpus.out
appl
appl.sir
aux_fields.F90
aux_func.F90
boundary.F90
calc_rhs_helper.F90
calc_rhs.F90
chease.F90
checkpoint.F90
circular.F90
collisions.F90
comm.F90
debug_output.F90
diag.F90
dummyperf.F90
eigenvalues.F90
erf.f90
erf.o
field_solve_kxky.F90
field_solve_xky.F90
fourier_acml.F90
fourier_essl.F90
fourier_fftw.F90
fourier_mkl.F90
time_scheme.F90
gene.F90
Outline
SIR Outli
Make Tar
SIR File
program: GENE (0/1790) (../src/gene.
userRegion: (608/1790) (../src/ger
call: GET_SYSTIME (32/32) (../sı
call: CALC_EXPLICIT_TIMESTEP
call: DIAG_NRG (32/32) (../src/g
call: DIAG_OMEGA (32/32) (../sı
call: DIAG_FIELD (32/32) (../src/
call: DIAG_MOM (32/32) (../src/
call: DIAG_VSP (32/32) (../src/g
call: DIAG_FIELD (32/32) (../src/
call: DIAG_MOM (32/32) (../src/
call: DIAG_VSP (32/32) (../src/g
call: CHECKPOINT_OUT (32/32)
call: CHECKPOINT_OUT (32/32)
call: RESCALE_G_1 (32/32) (../sı
call: DIAG_FIELD (32/32) (../src/
call: DIAG_MOM (32/32) (../src/
call: DIAG_VSP (32/32) (../src/g
call: DIAG_OMEGA (32/32) (../sı
subroutine: CALC_EXPLICIT_TIMESTE
loop: (517/517) (../src/time_schen
Problems
Console
Fortran Declaration
Periscope Properties View
Clustering Results View
Filter:
Search:
RE

```
   endif

   !----------------------------------
   !ALL OTHER STAGES
   !result is accumulated in k1
   do stage=2,rkstages
      if (stage.eq.2) then
         call nextstage_g(a_rk(stage)*dt,k1,g_1,tg)
      else
         call nextstage_g(a_rk(stage)*dt,k2,g_1,tg)
      endif

      call calc_rest(tg,emfields,f_)

      call CalFullRhs(f_,tg,emfields,k2,.false.)

      call add_ks(b_rk(stage)/b_rk(1),k2,k1)
   enddo

   !-----------------------------
   !RESULT OF TIMESTEPPING, new g
   call add_ks(b_rk(1)*dt,k1,g_1)

   !all other quantities
   call calc_rest(g_1,emfields,f_)

   Call perfoff
   DEBUG(1,"==== END   explicit_time_scheme ====")
 End Subroutine calc_explicit_timestep
 !===================== End Runge-Kutta ===============================
```

| Name | Filename | RFL | Severity | Region | Process | Thread |
|---|---|---|---|---|---|---|
| ▷ Stalls due to hardware page walker | | | 9.03 | Types Group | | |
| ▷ Stalls due to L2DTLB to L1DTLB transfer | | | 0.00 | Types Group | | |
| ▷ IA64 Pipeline Stall Cycles | | | 24.10 | Types Group | | |
| ▽ Stalls due to full store buffer | | | 0.20 | Types Group | | |
| Stalls due to full store buffer | ../src/time_scheme.F9( | 105 | 0.24 | LOOP_REGION | 4 | 0 |
| Stalls due to full store buffer | ../src/time_scheme.F9( | 105 | 0.21 | LOOP_REGION | 13 | 0 |

0 Loaded - 71 Shown - 1 Selected -
Sort: [Filename (FWD)]

Figure 5.3.: Overview of the Periscope's GUI for Eclipse

- Multiple criteria sorting algorithm - this allows the user to sort the presented data according to multiple different attributes of the performance properties. As an example, one can arrange the data first by severity of the problem, then by code region and file, and finally by process number to quickly identify the most critical hot-spots of the application.

- Complex categorization functionality - allows the users to organize the presented data hierarchically in smaller groups in order to concentrate their analysis only on, e.g., a particular code region or a specific performance metric such as cache misses or instructions-per-cycle (IPC).

- Search engine using regular expressions - most of the high-performance computing codes consist of many code regions and can potentially expose completely different runtime behavior when run in parallel on thousands of processors. As a result, a vast amount of performance properties will be generated which can be difficult to process. Consequently, the GUI provides a powerful search engine that allows users to highlight any particular attribute in the displayed performance data.
- Source code navigation - supports direct navigation from the displayed bottlenecks to their precise source location using any of the specialized source code editors of Eclipse. This way, one can easily see the source code context of a particular property and decide how critical that bottleneck is.

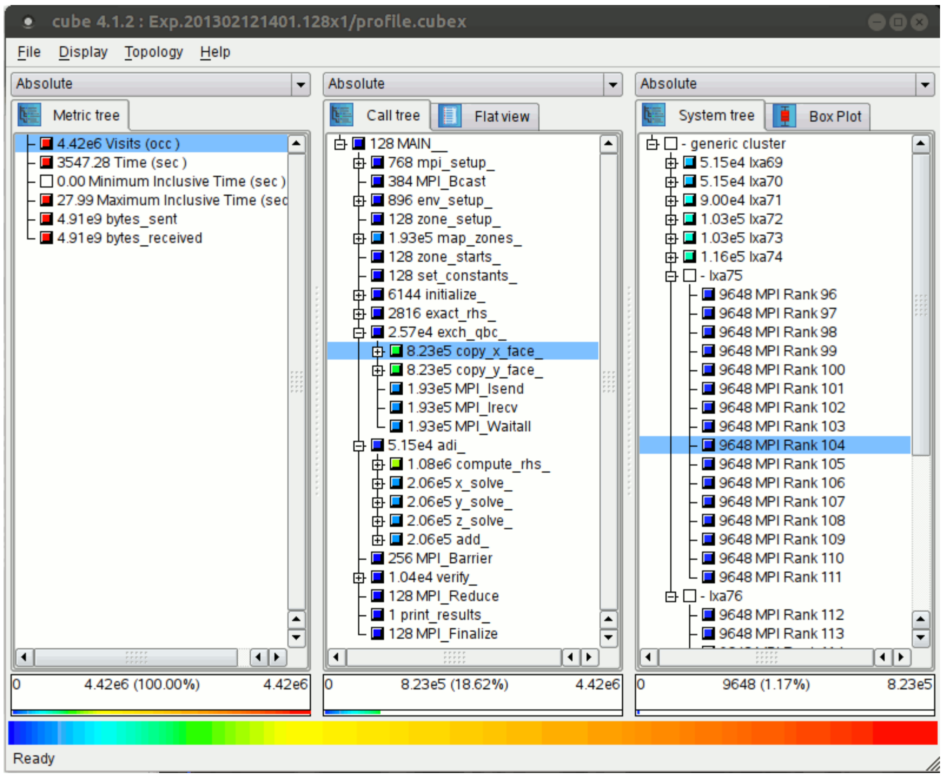
Furthermore, the graphical interface features an outline view of the code regions. It uses the Standardized Intermediate Representation (SIR) [185] of the analyzed application to display a tree-like structure of the source code. Each node in this hierarchy is additionally augmented with the total number of performance properties detected within it. As a result, the performance engineer can easily identify the most critical code regions and then quickly navigate to them.

Last but not least, another key feature of the Periscope's GUI is the multivariate statistical clustering support. Its design goal is to enhance the scalability of the UI and provide means of conducting large-scale performance analysis. Internally, the clustering functionality is based on the popular fuzzy K-means algorithm and can additionally use other algorithms provided by the WEKA Data Mining Workbench [83] from the University of Waikato in New Zealand. This feature can be used to automatically cluster code regions and detected performance properties and arrange them is groups having similar performance behavior. This way, less data has to be manually explored which once more results in a decreased time to solution and higher productivity of the performance engineers.

### 5.3.2. Scalasca

Scalasca [71] is a open-source performance analysis tool that is jointly developed by the Jülich Supercomputing Center and the German Research School for Simulation Sciences. Its design goals lay in the automatic analysis of large-scale, highly-parallel applications from industrial and academic environments. The tools supports MPI, OpenMP and hybrid programming paradigms where it can detect potential load-imbalance situations due to communication and synchronization bottlenecks like MPI Late Sender/Receiver or long wait states at global synchronization points. Moreover, it can also measure different hardware performance counters using the Performance Application Programming Interface (PAPI) [31]. When using Scalasca, the performance engineer can choose between the following two analysis modes in order to choose the level of detail about the performance behavior:

- Profiling - the advantage of this mode is that it is very fast and requires less storage. However, is provides only a high-level performance summary of the execution of an application without any load-imbalance analysis.
- Tracing - this mode is suitable for an in-depth study of application's behavior. It uses a post-mortem analysis approach based on the creation of a precise time-line of events. This approach requires, however, the collection and processing of large amount of performance data.



```
4.42e6 Visits (occ )
3547.28 Time (sec )
0.00 Minimum Inclusive Time (sec )
27.99 Maximum Inclusive Time (sec
4.91e9 bytes_sent
4.91e9 bytes_received
```

```
128 MAIN__
  768 mpi_setup_
  384 MPI_Bcast
  896 env_setup_
  128 zone_setup_
  1.93e5 map_zones_
  128 zone_starts_
  128 set_constants_
  6144 initialize_
  2816 exact_rhs_
  2.57e4 exch_qbc_
    8.23e5 copy_x_face_
    8.23e5 copy_y_face_
    1.93e5 MPI_Isend
    1.93e5 MPI_Irecv
    1.93e5 MPI_Waitall
  5.15e4 adi_
    1.08e6 compute_rhs_
    2.06e5 x_solve_
    2.06e5 y_solve_
    2.06e5 z_solve_
    2.06e5 add_
  256 MPI_Barrier
  1.04e4 verify_
  128 MPI_Reduce
  1 print_results_
  128 MPI_Finalize
```

```
- generic cluster
  5.15e4 lxa69
  5.15e4 lxa70
  9.00e4 lxa71
  1.03e5 lxa72
  1.03e5 lxa73
  1.16e5 lxa74
  - lxa75
    9648 MPI Rank 96
    9648 MPI Rank 97
    9648 MPI Rank 98
    9648 MPI Rank 99
    9648 MPI Rank 100
    9648 MPI Rank 101
    9648 MPI Rank 102
    9648 MPI Rank 103
    9648 MPI Rank 104
    9648 MPI Rank 105
    9648 MPI Rank 106
    9648 MPI Rank 107
    9648 MPI Rank 108
    9648 MPI Rank 109
    9648 MPI Rank 110
    9648 MPI Rank 111
  - lxa76
    9648 MPI Rank 112
    9648 MPI Rank 113
```

```
0   4.42e6 (100.00%)   4.42e6
0   8.23e5 (18.62%)    4.42e6
0   9648 (1.17%)       8.23e5
```

Figure 5.4.: Overview of the CUBE user interface

Scalasca utilizes a specialized, open-source data format called CUBE [72, 189] for the storage of collected performance data. Characteristic for this format is that it arranges the data in three-dimensional hierarchical structure. The first dimension represents the recorded performance metrics such as number of times a particular code region was run, its execution time, memory usage or the hardware counters. The second dimension shows the executed code regions in the form of a call-path. The last dimension describes the system component and gives information about the distribution of the performance data among nodes, processes and threads.

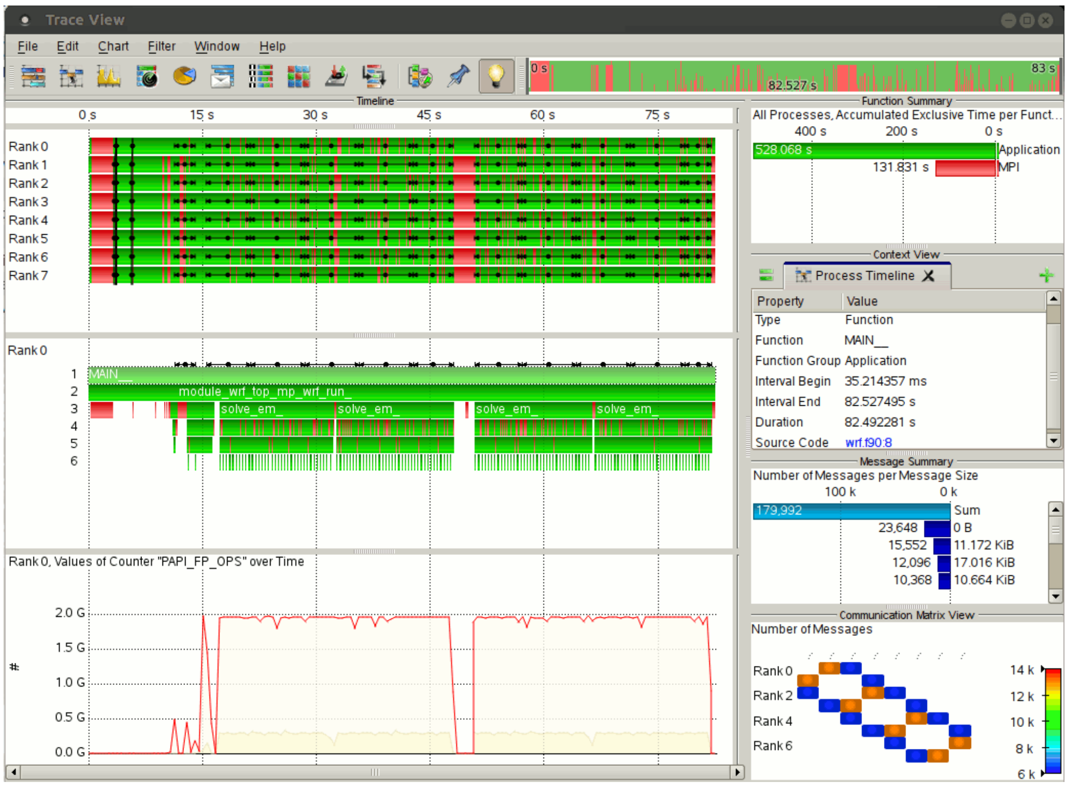
Moreover, this tool provides the performance engineer with a flexible QT-based graphical user interface for the interactive analysis of the collected monitoring data. As it can be seen in Figure 5.4, the GUI consists of three interlinked views that correspond, respectively, to the three dimensions of the CUBE data format. One can inspect the distribution of the performance data within the system dimension, i.e. node, process and thread, using the rightmost view. This is done by first selecting a node from the leftmost metrics view and then choosing one from the middle call-path tree. It is important to mention that when a node is not expanded, its value account not only for that particular node but also for all sub-nodes. For example, a collapsed data entry in the middle view shows the total inclusive value of the selected metric for the given code region and all its sub-regions.

### 5.3.3. Vampir

Vampir [120] is a commercial performance analysis tool that uses event trace profiling to support an in-depth analysis and visualization of the runtime behavior of parallel applications. It initially started as a spin-off from the VampirTrace open-source project of the Technische Universität Dresden and is now actively developed and distributed by GWT-TUD Ltd. Having its roots in the open-source community, the tool is still present in many publicly funded projects and actively collaborates in the development of many different open-source performance analysis tools and libraries such as Scalasca, TAU and Score-P.

Being a trace-based performance engineering tool, Vampir has the ability to generate highly detailed event time-lines from applications implemented using MPI, OpenMP or CUDA [142] programming models. The tool stores these event traces using an ASCII-based data format called Open Trace Format (OTF) [119]. By being open-source, this format allows for an easy data exchange between Vampir and other open-source analysis tools like CUBE and TAU. Recently, a new binary version of the tracing format named Open Trace Format Version 2 (OTF2) [59] was officially released. This version aims to greatly improve the scalability of tracing studies in order to meet the requirements of today's highly parallel systems.

In order to be able to process large-scale performance data, Vampir uses a client-server architecture. On the client side, it consists of a lightweight graphical user interface for the interactive exploration of performance data (see Figure 5.5). This GUI features a collection of optimized event trace processing algorithms and multiple highly customizable views. The most common display type in Vampir is the timeline chart which show the sequence of events in an execution trace. Another type is the statistical view which uses pie and bar charts to present metrics that specific to certain fine-grain measurement units like code regions, processes and threads.



| Property | Value |
|---|---|
| Type | Function |
| Function | MAIN__ |
| Function Group | Application |
| Interval Begin | 35.214357 ms |
| Interval End | 82.527495 s |
| Duration | 82.492281 s |
| Source Code | wrf.f90:8 |

Figure 5.5.: Vampir visualizing an event tracing study using Score-P

On the server side, the tool uses a scalable analysis engine to manipulate the high-volume tracing data in parallel. One of the advantages of this approach is that the collected performance data should not be copied locally to the user's machine which can otherwise be an extremely slow and expensive operation. Furthermore, the back-end analysis engine can use the full computational power of an HPC system for accelerating the data processing operations.

## 5.3.4. TAU and PerfDMF

Tuning and Analysis Utilities (TAU) [186, 187] is a parallel performance analysis system being jointly developed by the University of Oregon Performance Research Lab, the Jülich Supercomputing Research Center and the Advanced Computing division of the Los Alamos National Laboratory. It is designed not as a stand-alone tool but as an extendable framework and as such it is composed of many different modules covering most of the areas of performance engineering like instrumentation, monitoring, analysis and visualization. Additionally, the TAU group has recently started to extend the framework to support automatic tuning of parallel applications using machine learning techniques [38].

It has the ability investigate the performance behavior of applications written in Fortran, C, C++, Java, and Python. The framework design of TAU provides support for different types of application instrumentation. Depending on the developer's needs and the application under investigation, one can choose between the different instrumentation types:

- Source Instrumentation - using the Program Database Toolkit (PDT) [131], this approach provides TAU with a flexible and portable type of instrumentation. It is based on source-to-source modifications where measurement probes are directly inserted in the code of an application before it gets compiled. Additionally, this method allows performance engineers to manually fine-tune the source code instrumentation in order to limit the implicitly incurred measurement overhead.

- Binary Instrumentation - sometime the source code of an application is not publicly available and doing a source instrumentation is simply not possible. In order to measure the performance is these cases, TAU includes support for inserting the probes directly in the binary file using the binary rewriting techniques from the Dyninst [33] toolkit.

- Compiler Instrumentation - most modern compilers provide custom instrumentation support. This functionality does not require any extra source code manipulations or rewriting binary files as it automatically inserts measurement probes during the compilation of the target application.

- Library Wrapping Instrumentation - since many scientific applications depend on external libraries for calculation or data exchange, sometimes it is worth investigating their performance. This can be done using a technique called library interposition [50] where an intermediate wrapper library has to be generated (e.g. from its header files) and linked to the analyzed application. This library will first notify the monitoring infrastructure of the current call event and then it will execute the original library routine and return its results to the caller.

- Interpreter and Virtual Machine Instrumentation - the tool supports the automatic instrumentation of functions directly in both the Java Virtual Machine (JVM) using its profiling interface JVMPI [215] and the Python interpreter. This allows developers to do performance analysis of not only the typical for the HPC community C/C++ codes, but also Java and Python applications.

ParaProf [21] is a visualization and analysis tool for parallel performance data which was developed in the framework of the TAU project. Its design goal is to provide a flexible environment for analyzing and comparing large-scale performance engineering experiments. It supports different techniques for data representation like simple histogram analysis for displaying the distribution of measured metrics and sophisticated 3D visualization support for topologies and derived metrics. Additionally, it can aggregate results in qualitatively similar groups to ease their comprehension by a performance engineer.

TAUdb [88], which was originally called PerfDMF (Performance Data Management Framework), is another tool actively developed by the Oregon performance analysis group. It provides a generic performance database to store and organize both profile and trace performance data from different experiments. It is based on client-server architecture and is written in Java. The tool provides both C and Java APIs and an Eclipse integrated graphical user interface to allow performance engineers manage the data collected from performance experiments.

PerfExplorer [87] is TAU's effort in creating an integrated framework for parallel performance data mining and knowledge discovery. It integrates with the previously discussed TAUdb and provides a wide range of data mining operations that can be directly applied on large-scale performance data stored in the database. Moreover, using the clustering analysis feature of the framework, the data can be summarized by arranging it into logic groups exposing similar performance behavior. It supports the common k-means and hierarchical clustering algorithms and the quality of the grouping can be evaluated using either the Euclidean and Manhattan distance functions. Similarly like Periscope, these techniques help reduce the amount of performance data that has to be analyzed and thus help decrease the time to solution in performance engineering projects.

### 5.3.5. Score-P

Score-P [121, 146] is a new state-of-the-art performance measurement library. It is distributed under an open-source license and is being jointly developed by some of the leading HPC performance engineering groups in Europe and USA, i.e. Technische Universität München, Jülich Supercomputing Center, German Research School for Simulation Sciences, Technische Universität Dresden, and the University of Oregon Performance Research Lab.

On one hand, the library aims to provide a highly flexible performance monitoring environment to improve the productivity and interoperability of different analysis tools. This makes it possible for developers of such tools to concentrate on creating more comprehensive and productive analysis strategies instead of spending time to port their measurement infrastructures to different HPC environments and programming paradigms. On the other hand, using a common runtime monitoring library has the potential to decrease the learning overhead for applications developers and thus improve the acceptance of performance analysis tools by the community.

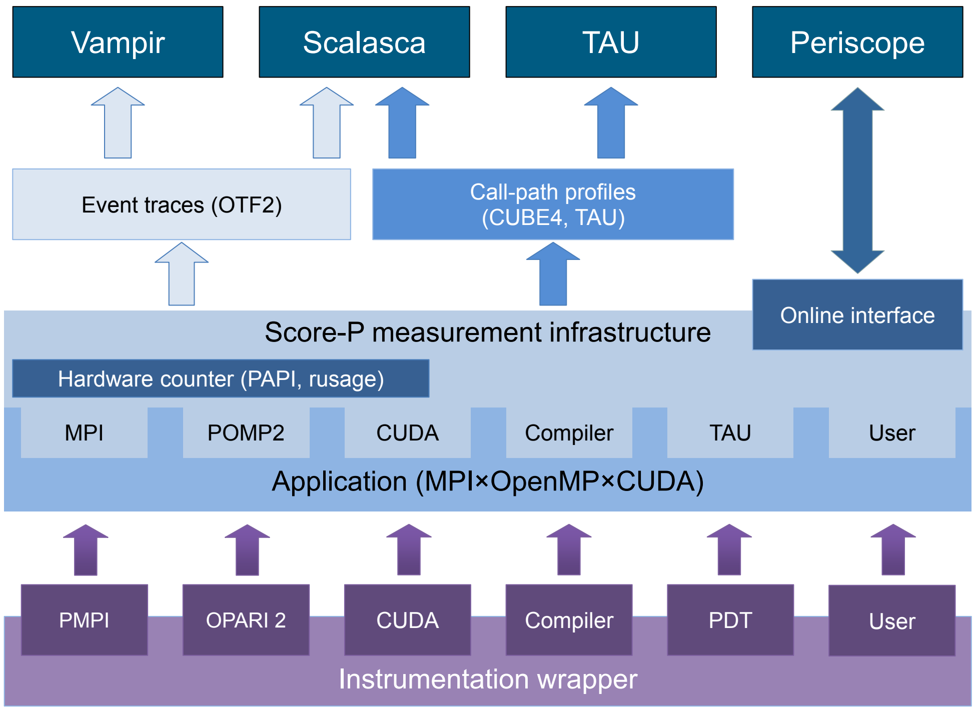


Figure 5.6.: Score-P: Architecture and integration overview [184]

Being designed as a shared instrumentation environment for the usage by many performance analysis tools, Score-P provides broad support for different hardware platforms and programming models, such as MPI, OpenMP, general-purpose graphics processing units (GPGPU) or a mixture of those. As it can be seen in Figure 5.6, the overall architecture of the library consists of three major layers: application instrumentation, monitoring infrastructure and storage of performance data. To be able to collect information about the runtime behavior of C/C++ and Fortran applications, Score-P currently supports six types of instrumentation:

- MPI using its profiling interface (PMPI) [81] and a wrapper library interposition technique;
- OpenMP using the OpenMP Pragma and Region Instrumentor Version 2 (OPARI2) [146] source-to-source instrumentation toolkit;
- GPGPUs using the NVIDIA's CUDA Profiling Tools Interface (CUPTI) [142];
- Compiler instrumentation of function calls using the performance monitoring support available in the majority of today's compilers;

- Automatic source code instrumentation using the PDT toolkit from TAU;
- Custom, fine-grained user instrumentation using a specialized API.

In order to improve the data exchange and interoperability between performance tools, Score-P was designed with the support of storing the collected measurement results in CUBE4, OTF2 or TAU snapshot file formats. As it was discussed before, these are open-source data storage formats that are supported by many performance analysis tools among which are Scalasca, Vampir and TAU. In addition, Score-P can use an efficient parallel I/O library called SIONlib [65] in order to improve the scalability of very large scale tracing-based performance engineering studies on today's and future highly parallel HPC systems.

Furthermore, an online sockets-based access interface is provided by the monitoring library. Using a Monitoring Request Interface (MRI) [116] specification, this interface can be used to control and reconfigure performance experiments at runtime and to provide easy access to the monitoring and data collection features of Score-P. For example, Periscope takes advantage of this functionality to conduct complex performance studies without requiring extra instrumentation utilities and monitoring infrastructure.

### 5.3.6. IBM HPCS and the BDE / SDE / SIE

The High Productivity Computing Systems (HPCS) Toolkit [47] from IBM is an attempt to create an integrated framework that not only automates the performance analysis process but also proposes possible solution guidelines and tries to automatically tune the monitored applications. Using plugable modules, new performance monitoring tools can be easily integrated to satisfy the specific needs of any performance engineer. The toolkit bases its data investigation routines on pre-defined abstract properties that are build upon multiple low-level metrics like hardware counters or time. The motivation behind this approach is to simplify the tuning process by directly identifying the possible causes for the detected bottlenecks instead of showing low-level performance data. Furthermore, the toolkit provides automatically an estimation for the expected performance improvement that can be achieved if any particular performance problem is removed.

The architecture of the IBM HPCS Toolkit is composed of four major components:

- Bottleneck Detection Engine (BDE) - module for parallel performance analysis using rules-based abstract bottleneck definitions.
- Solution Determination Engine (SDE) - module for processing the collected data in order to propose possible solutions to the detected bottlenecks. This module uses a predefined

knowledge database with tuning strategies from the IBM's experts in order to derive new optimization guidelines.

- Solution Implementation Engine (SIE) - module for transforming the source code of applications in order to automatically implement any of the proposed tuning strategies.
- Control GUI - user interface component that handles any required initialization steps such as selecting which code regions to be instrumented and starting a new performance experiment. Moreover, it manages the overall interoperability between the separate modules and is responsible for presenting the progress and the results to the performance engineer.

### 5.3.7. HPC Toolkit

HPC Toolkit [3] is a performance analysis framework developed by the Rice University (RICE) in Houston, USA. It consists of tools for collecting performance measurements without adding additional instrumentation. It analyzes directly the binaries of applications to identify the structure of the code and be able to correlate the measurement results with it. As a consequence of this, the tool is independent of any particular programming language. It uses different types of binary analysis techniques to handle both files compiled with debug symbols and optimized ones that do not include any source code information. The tool provides an binary analyzer that is capable of on-the-fly processing of machine code in order to compute call stack unwind strategies to map the detected performance bottlenecks. This approach combined with tool's sampling-based performance measurement approach allows the performance measurement perturbation to be minimized.

PerfExpert [34] is a performance measurement and optimization tool which developed by the Texas Advanced Computing Center. It aims at capturing the hardware architecture, the system software and the compiler knowledge necessary for an effective performance engineering process. In the background, this tool interfaces with the HPCToolkit to execute a structured sequence of performance measurements in order to collect low-level runtime data. After analyzing the gathered results, it tries to identify any potential bottlenecks from any of the following six categories: data memory accesses, instruction memory accesses, floating-point operations, branch instructions, data Translation Lookaside Buffer (TLB) accesses, and instruction TLB accesses. Furthermore, the performance engineer can use an online service called AutoSCOPE [190] to generate a list of possible optimizations guidelines for any of the detected bottlenecks. For example, this information can be as simple as providing different set of compiler flags or can include full source code modifications.

## 5.4. Performance Tuning Libraries and Frameworks

With the beginning of the multicore era, squeezing out the full potential of modern processors has become a very challenging task. The main task for many application developers has shifted from pure software implementation to performance tuning for current and emerging parallel systems. This activity deals with the extensive search for the most optimal mixture of a runtime configuration and code modifications. However, this inevitably results in an enormous search space which further complicates the whole performance tuning process. As a result, many research groups has devoted their resources for the sake of improving the area of automatic tuning in the last years and, as it will be seen in the next sub-sections, many important ideas have been successfully identified.

### 5.4.1. Automatic Performance Tuning HPC Libraries

As an attempt to accelerate the development of highly optimized applications, a set of special purpose highly optimized libraries were developed for many application areas. For example, the Automatically Tuned Linear Algebra Software (ATLAS) [45] supports application developers in creating numerically intensive computational codes. Its design goal is to automatically generate optimized versions of the popular Basic Linear Algebra Subroutines (BLAS) [125] kernels for many different hardware architectures. Similarly, Fastest Fourier Transform in the West (FFTW) [64] is another common library for computing discrete Fourier transforms on different HPC systems. Due to its optimized implementation, simulation codes using FFTW will perform very fast on the majority of today's architectures without requiring additional source code modifications. Other popular examples of auto-tuning libraries are presented in [106, 216, 170].

### 5.4.2. Search-based Performance Tuning Tools

The growing diversity of application areas for parallel computing requires, however, a more general auto-tuning strategy. Thus, a substantial research has been done in other application-independent directions. One of those deals with the automatic search for the most effective combination of compiler flags for a specific platform. There are two main groups of performance tuning tools that implement such an approach: those that implement brute-force search techniques and the ones that use machine learning algorithms. There has been much work in the first category as described in [205, 85].

These tools typically apply many different combinations of compiler flags and evaluate their suitability based on actively running the parallel applications and monitoring the achieved effect on the runtime. Due to the extremely large search space created by all possible compiler configurations, specialized search algorithms [163, 164] were developed to accelerate this process. However, most of the brute-force automatic tuning tools are still very slow and expensive to utilize.

As mentioned previously, the second group of compiler-based auto-tuners applies a different strategy to identify the fastest combination of flags. They initially build up knowledge about the program's behavior and then use machine learning techniques to predict the optimal configuration [127] for a specific hardware architecture. Internally, these tools automatically create analytic models of the underlying system that map compiler flags to a certain performance behavior by running tiny, predefined benchmark codes. These models can then be applied to real-world simulations codes to guide the tuning process.

There also exists research targeting the creation of a self-optimizing compilers that automatically learn the best optimization heuristics based on the behavior of the underlying platform. An example for this is the cTuning Compiler Collection [66] that integrates the Continuous Collective Compilation Framework (CCC) [68], an infrastructure of collaborating tuning agents, and the MILEPOST GCC [67] self-tuning compiler.

### 5.4.3. Automatic Performance Tuning Frameworks

Among the examples of auto-tuning frameworks is Autopilot [175]. It is a tuning framework that provides an integrated environment for performance monitoring and dynamic tuning of heterogeneous computational applications based on closed-loop control. It uses distributed sensors to extract qualitative and quantitative performance data from the investigated applications. This data is processed by distributed actuators and the preliminary performance benchmark is reported to the application developer.

Active Harmony [44] is a runtime parameter optimization tool that can automatically adapt the parameters that are most critical for the overall performance of an application. It can be used to tune parameters such as the size of buffers or the selection of a specific implementation of an algorithm, e.g., heap or quick sort. Furthermore, the framework can try to improve the performance based on the observed historical performance data during only a single execution.

Parallel Active Harmony [203] is a combination of Active Harmony and the CHiLL [40] code transformation framework. While monitoring the performance of the application, the system investigates multiple dynamically generated versions of code regions having the strongest overall performance impact. Characteristic for the framework is that the performance of these code regions is evaluated in parallel directly on the target HPC system and the results are processed by a parallel search algorithm. At the end of the search process, the code version with best performance will be integrated into the optimized version of the application.

The work from Nelson et al. [156], in contrast to Active Harmony, uses a model-guided empirical optimization approach that interacts with the programmer to get high-level models of the impact of parameter values. These models are then used by the system to guide the search for optimization parameters.

Periscope Tuning Framework (PTF) [147] is an extension to the Periscope toolkit for automatic tuning of both serial and parallel HPC applications with respect to performance, energy efficiency and suitability for GPU-based accelerators. It is currently being developed by Technische Universität München, Leibniz Supercomputing Center (LRZ), CAPS Entreprise, Universitat Autònoma de Barcelona, University of Galway (ICHEC) and Universität Wien. It uses the Periscope performance analysis tool together with a set of external tuning plug-ins to automatically assess different optimization opportunities and search on-the-fly for the one having the best performance in the target parallel environment. The target areas for automatic tuning include single-core performance tuning using compiler flags, MPI optimization by employing different environment parameters, tuning of GPU-based computational kernels utilizing HMPP (Hybrid Multicore Parallel Programming) and OpenCL (Open Computing Language) and adapting the runtime of applications for increased energy efficiency. After every successful performance engineering study, the framework generates a comprehensive report with optimization guidelines that effectively build the foundations for creating a highly-tuned, platform-specific version of the analyzed application.

## 5.5. Eclipse and the Parallel Tools Platform

Parallel simulation codes tend to have complex architectures and are often composed of thousands of lines distributed in many different files. This makes the usage of simple text editors like `vi` for the development quite a challenge. That is why, different supportive environments are developed. These so called integrated development environments (IDE) aim to greatly improve the implementation and maintenance processes of big software codes and allow the developers to concentrate on what matters most - the creation of new products.

### 5.5.1. Eclipse Integrated Development Environment

Eclipse [56] is an open-source, extensible, vendor-neutral platform that includes plenty of development features. In the course of time, it has become the preferred open-source IDE used by thousands of developers world-wide. The supported programming languages range from C/C++, through Java, till scripting languages like Python and LaTeX. Due to its open-source license, the platform has worldwide redistribution rights and can be modified and extended by anyone. There is a big community of users that constantly extends the functionality of Eclipse, thus making it even more powerful.

Since Eclipse is entirely implemented in Java, it is highly portable. It is available for all major versions of Linux, Windows and Mac OS X. Furthermore, the IDE has support for development of both local and remote projects, thus being suitable for many application areas ranging from embedded development to high-performance computing. The base part of the IDE is consisted of a simple runtime system and a set of platform-dependent libraries. This provides basic user interface and internal management infrastructure. Everything else comes in the form of additional plug-ins that contribute new functionalities to the platform.

### 5.5.2. Parallel Tools Platform (PTP)

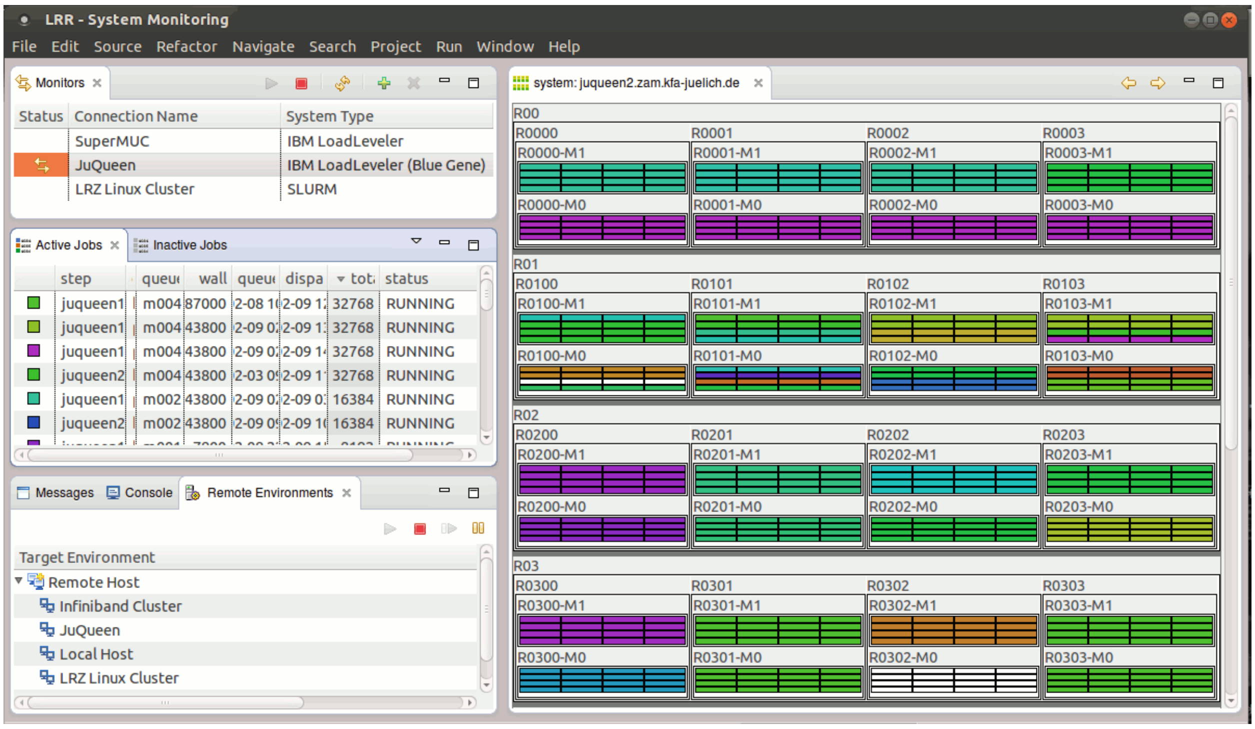
The Eclipse Parallel Tools Platform (PTP) [6, 222] is another extension for the IDE which provides a higher-level abstraction of the underlying high-performance computational resources. Recently, it was released within a special version of Eclipse that is specially targeted to the development of complex parallel source codes.

One of its biggest advantages is the abstraction of the underlying parallel computing resources. This enables easy usage of the systems by providing a convenient way for submitting and monitoring both parallel and sequential jobs (see Figure 5.7). It supports remote C/C++ projects as if the files were on the local machine and another special type of internally synchronized projects. This provides all the normal code refactoring and code assist functionality of Eclipse without having to deal with multiple copied of the same software code.

#### Remote Tools

In the field of high performance computing, computational resources are often dedicated governmental systems that are isolated from the public internet using strict firewalls. They can be accessed only from special gateway machines using only secure protocols like SSH. This limits the applicability of standard software development tools and so makes the overall development process tedious.

In order to ease the interaction with such systems, PTP implements a set of remote service providers among which is the Remote Tools extension. The goal of these providers is to abstract the network connections between local and remote hosts and provide a high-level API. Using this functionality, Eclipse and the other PTP components can transparently access remote file and operating systems as if they were local resources and implement their standard features in a platform-independent way.


LRR - System Monitoring
File Edit Source Refactor Navigate Search Project Run Window Help
Monitors
Active Jobs
Inactive Jobs
Messages Console Remote Environments
system: juqueen2.zam.kfa-juelich.de
R00 R0000 R0001 R0002 R0003 R0000-M1 R0001-M1 R0002-M1 R0003-M1 R0000-M0 R0001-M0 R0002-M0 R0003-M0
R01 R0100 R0101 R0102 R0103 R0100-M1 R0101-M1 R0102-M1 R0103-M1 R0100-M0 R0101-M0 R0102-M0 R0103-M0
R02 R0200 R0201 R0202 R0203 R0200-M1 R0201-M1 R0202-M1 R0203-M1 R0200-M0 R0201-M0 R0202-M0 R0203-M0
R03 R0300 R0301 R0302 R0303 R0300-M1 R0301-M1 R0302-M1 R0303-M1 R0300-M0 R0301-M0 R0302-M0 R0303-M0

| Status | Connection Name | System Type |
|---|---|---|
| | SuperMUC | IBM LoadLeveler |
| ⇆ | JuQueen | IBM LoadLeveler (Blue Gene) |
| | LRZ Linux Cluster | SLURM |

| | step | queu | wall | queu | dispa | tot | status |
|---|---|---|---|---|---|---|---|
| 🟩 | juqueen1 | m004 | 87000 | 2-08 1 | 2-09 1 | 32768 | RUNNING |
| 🟩 | juqueen1 | m004 | 43800 | 2-09 0 | 2-09 1 | 32768 | RUNNING |
| 🟪 | juqueen1 | m004 | 43800 | 2-09 0 | 2-09 1 | 32768 | RUNNING |
| 🟩 | juqueen2 | m004 | 43800 | 2-03 0 | 2-09 1 | 32768 | RUNNING |
| 🟩 | juqueen1 | m002 | 43800 | 2-09 0 | 2-09 0 | 16384 | RUNNING |
| 🟦 | juqueen2 | m002 | 43800 | 2-09 0 | 2-09 1 | 16384 | RUNNING |

Target Environment
- Remote Host
  - Infiniband Cluster
  - JuQueen
  - Local Host
  - LRZ Linux Cluster

Figure 5.7.: Parallel Tools Platform monitoring a remote system

### JAXB Resource Manager

A resource manager plays central part in the design of the Parallel Tools Platform. It interfaces with available remote system providers in order to transparently access remote machines. It then cooperates with the underlying parallel environment of these systems to start user applications and monitor their execution status.

In the past, PTP included multiple separate resource managers to be able to interact with the different parallel environments that were used on HPC systems. It had support for the majority of batch schedulers and MPI implementations available at that time. However, each resource manager was implemented in its own, very specific way and depended on additional modules to be installed on the remote systems. Furthermore, there was no support for generic configuration of the provided managers in order to adapt them to the specific systems of different computing centers.

As a result, the developers of PTP have recently decided to rewrite the whole resource management infrastructure and provide a generic implementation called the JAXB Resource Manager. It uses an XML based format to describe the low-level interface with the parallel environment and its requirements. This allows developers to not only create their own fine-tuned resource orchestrators, but also to adapt existing ones to their specific requirements without writing extra code. Moreover, since this implementation does not depend on the existence of any external components, it is highly portable and much easier to setup.

#### Synchronized Projects

The goal of the Synchronized Projects feature of PTP is to allow easy and semi-transparent remote development. This is achieved by internally mirroring local files and folders to the remote machines and later on synchronizing them on special events such as saving modified files, recompiling or starting the developed application or when a full re-synchronization is requested by the user. Moreover, building of the project will be done automatically on the remote system in order to eliminate extra manual steps.

The biggest advantage of this approach is that no constant network connection is required during the development. This allows programmers to continue working when, for example, there is no network access or the remote target machine is not reachable due to maintenance. Furthermore, the synchronized projects can be configured to mirror and build the code simultaneously on multiple remote machine which can greatly ease the process of cross-platform development.

#### External Tools Framework (ETFw)

The aim of the External Tools Framework (ETFw) is the integration of performance analysis or visualization tools in the PTP framework. Using a specialized XML format, this extension allows dynamic creation of UI components, i.e. check- and comboboxes, text fields and labels and file system selection dialogs, for the configuration and execution of such tools within the Eclipse IDE.

Furthermore, users can use this format to specify a sequence of commands for code instrumentation, monitoring of applications and post-processing of performance results in order to automate the execution of a simple performance analysis study. The framework will then assemble these in a workflow model and will execute them in the specified order on behalf of the user.

#### Parallel Language Development Tools (PLDT)

Parallel Language Development Tools (PLDT) are another productivity feature of PTP that targets the automatic detection of language artifacts, i.e. function calls or code constructs, and common parallel programming problems. In case of OpenMP applications, the developer can easily find misplaced parallel regions or concurrency problems. Moreover, static barrier analysis is provided for the automatic detection of potential deadlocks in MPI codes.

Furthermore, the PLDT feature provides support for code snippets and content-sensitive assistance for not only MPI and OpenMP programming models, but also for UPC [36], OpenACC [141] and OpenSHMEM [39] codes. This effectively lowers the barrier to entry for inexperienced developers and thus eases the adoption of parallel programming by a broader community.

# Part II.

# PAThWay to Performance Analysis and Tuning Workflows

Chapter 6

# Performance Engineering Workflows

**Contents**

**6.1. Performance Engineering Workflows . . . . . . . . . . . . . . . . . . . . . . . . . . 71**
**6.2. Workflow for Scalability Analysis of Parallel Applications . . . . . . . . . . . . . 72**
**6.3. Workflow for Cross-Platform Performance Analysis . . . . . . . . . . . . . . . . . . 74**
**6.4. Workflow for Code Migration and Tuning on GPGPUs . . . . . . . . . . . . . . . 76**
**6.5. Summary and Requirements Overview . . . . . . . . . . . . . . . . . . . . . . . . . . 78**

## 6.1. Performance Engineering Workflows

In the field of performance engineering , there are many highly iterative, long-running processes which could easily benefit for a higher-degree of automation. However, these are often performed manually by the involved users without the support of software systems. Furthermore, even though the majority of these processes are very similar in nature, no form of standardization or a set of best-practices currently exists.

In order to address these topics, this thesis proposes an generic approach of automating and standardizing the typical operations carried out by today's performance engineers. Before going into the details about this proposal and the implementation of PAThWay, however, three of the most popular performance engineering workflows will be presented in the following sections. Their high-level definitions will be discussed in order to provide a broader overview of the involved process steps and to extract their inherent automation requirements. In later chapters, these will be thoroughly analyzed and mapped to the architectural decisions behind PAThWay.

## 6.2. Workflow for Scalability Analysis of Parallel Applications

One of the very fundamental performance engineering processes is the analysis of the scalability of parallel applications. Its aim is to the investigate the effect of adding more computational resources on the runtime behavior of algorithms. By running the target application multiple times with a varying number of processors, performance data is collected in order to estimate the change of the overall execution time of the code and the utilization of the underlying hardware.

There are two different types of scalability studies: *strong* and *weak* scalability analyses. The main difference between them is the change of the size of input data. One one hand, strong scaling is when the problem size is fixed while the number of processors is being increased. As a consequence, the amount of computational data per process decreases, allowing the application to reach a solution of the problem faster. Using such an approach, one can investigate the minimum time required to solve a particular problem on a specific system or to find the optimal number of processors that achieve maximum computational efficiency for the given problem. On the other hand, weak scaling is when the problem size increases together with the growing number of processors. This type of scalability study explores the suitability of an algorithm to solve bigger problems using more computational resources for a fix amount of time.

Parallel speedup (6.1) and efficiency (6.2) are the most important metrics when investigating the scalability of an application. These represent the relation between sequential and parallel execution behavior and are calculated as

$$S_p = \frac{T_1}{T_p} = \frac{1}{f_p/p + (1 - f_p) + O_p * p} \tag{6.1}$$

$$E_p = \frac{S_p}{p} = \frac{T_1}{p * T_p} \tag{6.2}$$

where $T_1$ is the overall execution time using 1 processor and $T_p$ is the running time of the code on $p$ processors. According to *Amdahl's law* [86], the maximum speedup of any parallel application is bounded by the fraction of time needed for executing the parallel section of the code, i.e. $f_p$, and the management overhead $O_p$ incurred by parallel execution system. In the case of embarrassingly parallel algorithms, this fraction is close to 1 which leads to speedup equal to the number of processors in use. However, due to the fact that the execution time of all sequential regions remains constant during a parallel run, even small changes in $f_p$ can have a strong negative effect on the speedup of parallel applications.

The scalability analysis process was selected because of its uncomplicated definition and vast popularity among the HPC community. As it can be seen from the high-level workflow model in Figure 6.1, this process consist of multiple interconnected components in order to provide transparent interface between local and remote systems and enable automatic execution of experiments.

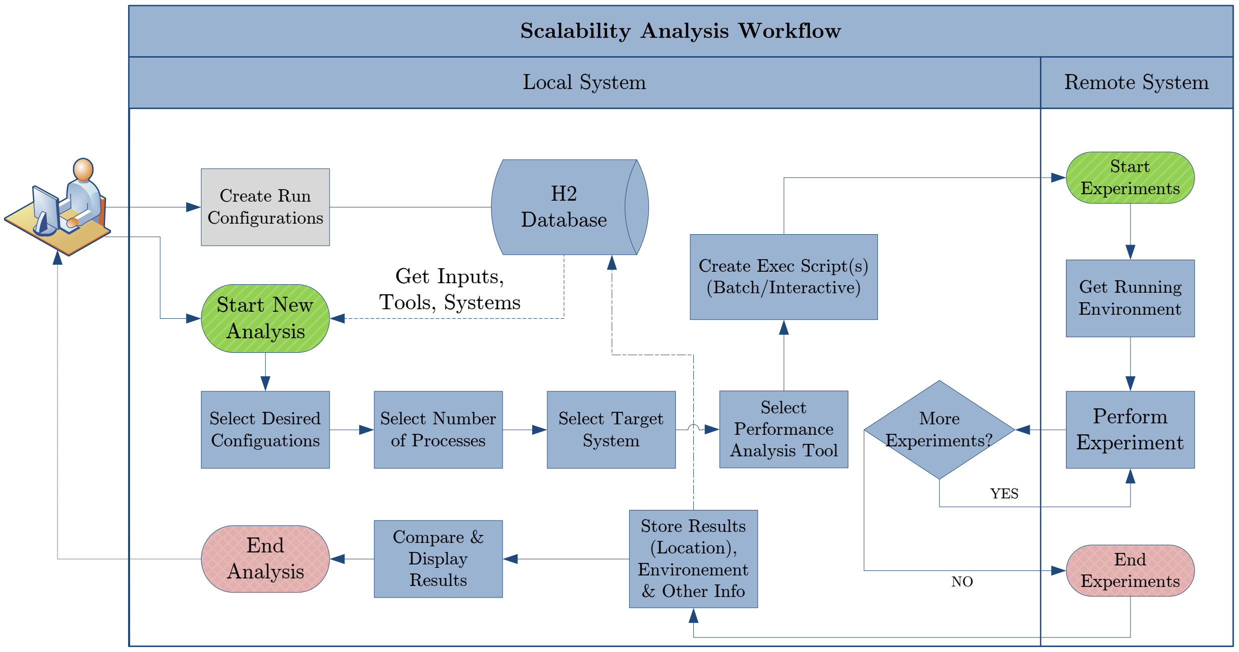

Scalability Analysis Workflow
Local System
Remote System
Create Run Configurations
H2 Database
Start New Analysis
Get Inputs, Tools, Systems
Create Exec Script(s) (Batch/Interactive)
Start Experiments
Get Running Environment
Select Desired Configuations
Select Number of Processes
Select Target System
Select Performance Analysis Tool
More Experiments?
Perform Experiment
YES
NO
End Analysis
Compare & Display Results
Store Results (Location), Environement & Other Info
End Experiments

Figure 6.1.: Scalability Analysis Workflow Model

First, there should be a way to interact with the performance engineer in order to configure the whole scalability study. This includes the selection of an application to be analyzed and its runtime parameters including environment variables and external modules. Furthermore, the desired range for the number of MPI processes and OpenMP threads and the target parallel system should be specified. Even though monitoring the execution of applications can be done simply using system's wall time, it is advisable that users apply performance analysis tools to be able to gather a deeper insight about the application's behavior. Consequently, the workflow model should be able to accommodate the selection of preconfigured performance analysis tools as part of the interactive configuration.

Second, the workflow should be able to automatically generate experiments based on the information provided by the user. This will include implicit expansion of the given range of MPI processes and OpenMP threads in separate execution units and adapting the settings of the application to the selected HPC environment. It is important that this is done in a platform-independent way using a format suitable for executing on the chosen HPC machine.

Third, the workflow model should define an interface to a job submission and monitoring system that will handle the low-level communication and execution tasks on behalf of the user. This system should be able to autonomously execute experiments as configured by the performance engineer and record the runtime environment of each job simultaneously. It should not require the user to manually establish a network connection to the remote machine or create any scripts. After submitting an experiment to the target HPC system, its execution state has to be closely monitored and reported back to the user. Ideally, this monitoring phase should be able to restart automatically in case of a network failure or a shutdown of the local machine.

Fourth, once all sub-experiments of a scalability study are finished, meta-information about their execution such as standard output and error, location of performance results and execution environment has to be collected and presented to the user. In addition, this information has to be stored in a persistent manner. This will not only increase the transparency and the traceability of the overall performance engineering process but will also allow the user to go back and exactly repeat past experiments at any future time. For example, one might want to test previous code changes on a new compiler or with a new kernel and evaluate if they are still beneficial for the application's runtime or one might need to recreate missing experiment results for an article.

Finally, the workflow model should provide a way of extracting key performance metrics about the scalability of the analyzed application and be able to aggregate the results in a format suitable for manual exploration. This step can provide the performance engineer with a set of instructions on how to access and explore the collected performance data or it can go one step further and fully automate the post-processing activity. The later case can, for example, include an automatic generation of scalability plots for all analyzed code regions or for the application as a whole. Alternatively, it can also display an graphical user interface where the performance engineer can interactively select different code regions and study the evolution of their behavior with changing number of processors.

## 6.3. Workflow for Cross-Platform Performance Analysis

At the present time, there are multiple high-performance computing vendors world-wide. The big ones like IBM and Cray provide full end-to-end products that include both proprietary hardware and software components, user training and support and normally target big computing centers. Others, much smaller companies, offer only off-the-shelf hardware and customization services and typically work with small businesses and research groups. However, they can be sometimes also found at governmental supercomputing sites. As a consequence, there is a very broad spectrum of computing environments in the HPC area with each one having its advantages and disadvantages and being suitable for running different types of applications.

That is why, another very common process in the field of performance engineering deals the analysis and the comparison of different hardware architectures and runtime execution environments. The goal of this activity is to identify the most suitable parallel platform for a given code so that the best resource utilization can be achieved. Typically, this process requires the execution of the target application on multiple machines using selected input configurations in order to collect key performance metrics about its runtime behavior. Based on these data, the suitability of the underlying computing architecture can then be evaluated.

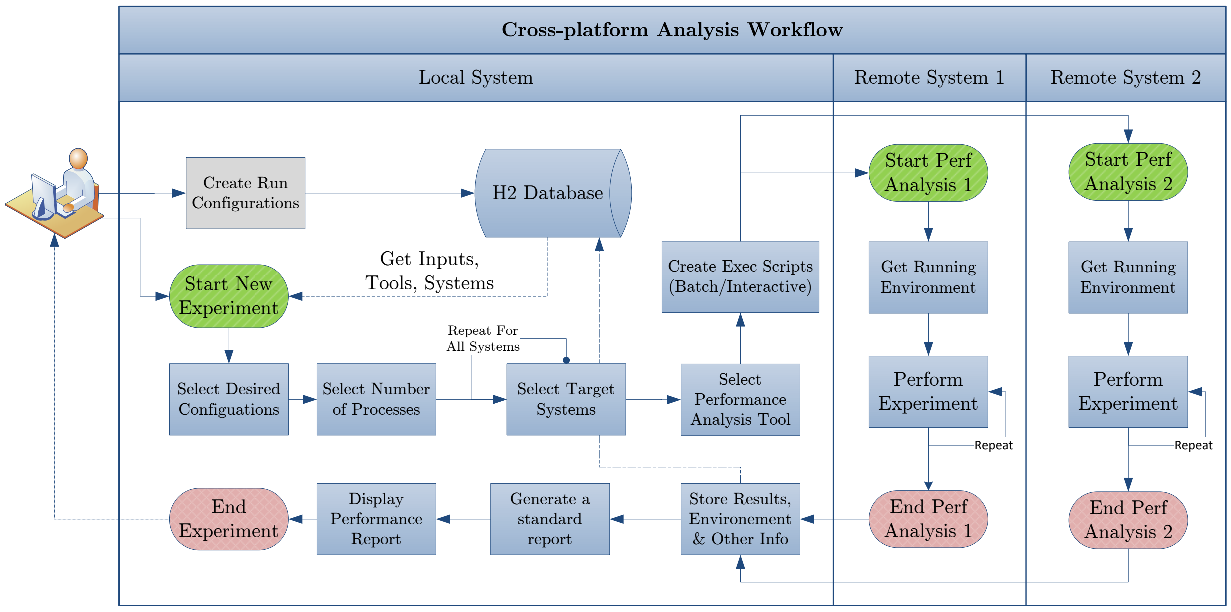


Figure 6.2.: Cross-Platform Analysis Workflow Model

Similarly to the scalability analysis workflow presented in Chapter 6.2, this performance engineering process starts with the interactive selection of a parallel application and the configuration of the execution environment such as number of MPI processes or OpenMP threads (see Figure 6.2). In contrast to the previous workflow, however, here the user has to choose not only a single parallel system but a whole collection of them. Then, just as the other one, the setup phase finishes with the selection of a performance analysis tool and the generation of all required experiments.

Once the whole cross-platform study is configured, the underlying job submission and monitoring system should dispatch the generated experiments to all selected computational resources without any manual intervention. In order to achieve traceability of the overall process, this workflow has to also store persistently meta-information about the runtime environment of each experiment together with all collected results.

As discussed previously, there are many different parallel systems that one can use. Each one of them has its own architecture and it supports a different set of distinct features such as different batch systems, directory structures, etc. When it comes to monitoring the runtime behavior of parallel applications, the disparity between the machines becomes even bigger. This makes it challenging to choose a common set of performance metrics that can be measured across all the machines and then consistently compared with each other. What is more, this step might require the extension of the applied performance tools to support the same set of measurements independently of the underlying architectures. From user's perspective, all these prerequisites should ideally stay hidden and instead a standardized report about the results should be generated as an outcome of this performance engineering process.

In conclusion, the resulting report about the performance characteristics of the code executed on the different systems should be manually explored by the user. If the performance engineer wants to get a deeper insight about each separate machine, access to the collected results and environment settings should also be easily possible. This way, a convenient environment for analyzing the suitability of HPC architectures for a given application can be achieved.

## 6.4. Workflow for Code Migration and Tuning on GPGPUs

In the last couple of years, a new computing architecture has become popular in the high-end HPC market. Coming from the field of graphics processing where a very large amount of relatively inexpensive, independent operations have to be executed, this architecture is based on many lightweight, on-chip CPU cores which can work simultaneously. This creates a big opportunity for parallelism which constantly attracts many new application developers and fosters further research and development of this technology.

Consequently, a new type of a performance engineering workflow is gaining popularity among the HPC community. It deals with the migration and optimization of existing source codes to these highly-parallel architectures. Characteristic for this type of devices is that they do not support standard programming paradigms such as MPI or OpenMP but rather require specialized frameworks such as CUDA [142] or OpenCL [154]. Unfortunately, these specifics eventually hinder the adoption of the technology by a broader community.

However, a new research initiative aiming at developing a generic programming interface for such accelerators has been recently started. It is called OpenACC [141] and its goal is to create a programming model that can be used to implement high-level applications supporting both CPUs and GPGPUs. Due to its similarity with OpenMP, existing parallel applications can be easily converted in order to take advantage of this technology. Nevertheless, this migration process still requires multiple steps in order to for the code to squeeze the maximum performance out of the

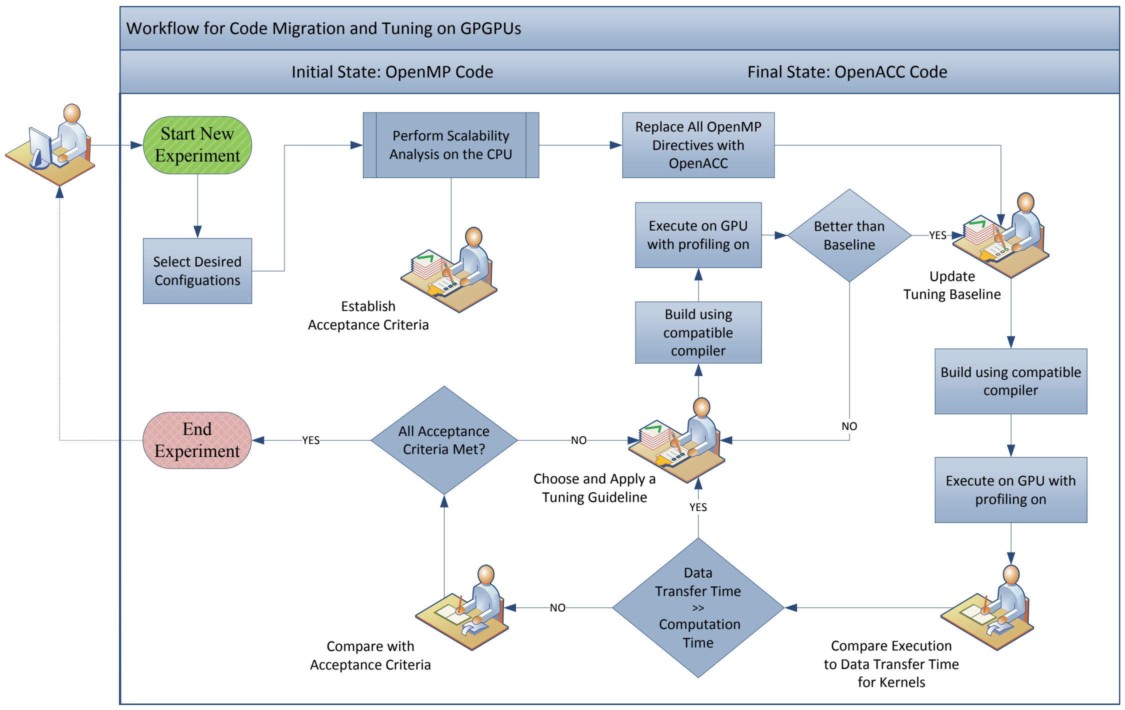

Workflow for Code Migration and Tuning on GPGPUs
Initial State: OpenMP Code
Final State: OpenACC Code
Start New Experiment
Select Desired Configuations
Perform Scalability Analysis on the CPU
Establish Acceptance Criteria
Replace All OpenMP Directives with OpenACC
Execute on GPU with profiling on
Better than Baseline
YES
Update Tuning Baseline
Build using compatible compiler
Build using compatible compiler
NO
End Experiment
YES
All Acceptance Criteria Met?
NO
Choose and Apply a Tuning Guideline
YES
Execute on GPU with profiling on
Data Transfer Time >> Computation Time
NO
Compare with Acceptance Criteria
Compare Execution to Data Transfer Time for Kernels

Figure 6.3.: Workflow Model for Code Migration and Tuning on GPGPUs

hardware and, as such, is another good candidate for automation.

In contrast to the previous two workflows, this one describes a semi-automatic process. As it can be seen in Figure 6.3, it is a mixture of multiple manual activities, automated steps and a complex sub-process which poses very different challenges to the automation environment. For example, manual tasks such as establishing a suitable acceptance criteria or implementing code modifications can span over multiple days and this should be handled by the workflow engine.

Additionally, it should be possible to integrate other workflow models as single steps in order to encapsulate separate execution units and decrease the complexity of the resulting worklflow. A study of the scalability of the code on a CPU is normally the first task in the migration process as it is crucial for the performance engineer to understand the runtime behavior of the application before proceeding with any further steps. This is, however, often a long-running process having its own workflow model.

Last but not least, the performance engineer should be able to access an existing knowledge database with OpenACC tuning guidelines as part of the optimization process. This can be achieved by interfacing with specialized automatic tuning systems like the Periscope Tuning Framework (see Chapter 5.4.2) or it can be as simple as embedding expert hints directly in the workflow.

## 6.5. Summary and Requirements Overview

In the previous sections, three of the most common performance engineering workflows were presented and the inherent requirements they impose on a system aiming at their automation were discussed. Since these requirements serve as the basis for the architectural design of PAThWay, they are once more shortly summarized in the following points:

- An interactive interface for configuring new performance engineering studies should be provided. It should enable the selection of applications to be used within a workflow together with their runtime settings, target HPC systems, execution settings of the underlying parallel runtime environments (e.g. number of MPI processes and OpenMP threads) and others.
- Generic access to performance analysis tools should be enabled as part of the setup phase. This should encapsulate their runtime dependencies and settings so that users can analyze their applications without having to learn in-detail the provided analysis tools.
- The system should be able to autonomously generate complete experiments based on the information provided by the user during the configuration step.
- A job submission and monitoring system should be included to handle the low-level communication and execution of experiments on behalf of the user. Furthermore, it should be able to monitor the execution state of experiments and report that back to the user.
- The system should be able to autonomously collect meta-information about the execution of performance engineering experiments such as standard output and error, location of performance results and execution environment and to persistently store that for a later reference.
- As part of a workflow, there should be a way of automatically extracting and aggregating key performance metrics about performance experiments, e.g. execution time, communication overhead, etc. These can then be summarized in a standardized report about the results as an outcome of a performance engineering process.
- Access to existing knowledge databases or utilization of external tuning frameworks from within workflows should be possible to ease the provision and selection tuning guidelines.
- Due to the semi-automatic nature of some performance engineering tasks, long-running manual activities should be supported by the system.
- The system should enable the integration of complete process models as single workflow steps in order to modularize and better encapsulate execution logic and thus decrease the complexity of the resulting workflow models.

Chapter 7

# Architecture for Automation of Performance Engineering Processes

**Contents**

**7.1. Architecture for Automation of Performance Engineering Processes . . . . . . . . 79**
**7.2. PAThWay Graphical User Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 80**
**7.3. Logic Implementation Layer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81**
**7.4. Supportive Modules Layer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83**

## 7.1. Architecture for Automation of Performance Engineering Processes

PAThWay is deliberately designed using a multilayered software architecture in order to overcome the challenges imposed by the high complexity of such a cross-platform tool. Even though many of the software components of PAThWay are executed locally on the user's machine, the framework still needs to be able to interact with many different HPC systems where each has its own specific runtime requirements and access restrictions. Moreover, due to the extremely fast evolution of the HPC field, it has to be easily extendable in order to be able to adapt to any forthcoming runtime configurations and underlying execution systems without considerable efforts from the side of either the involved developers or the users.

The architecture of PAThWay can be roughly separated in three interconnected layers which are displayed using different colors in Figure 7.1. The topmost layer, having blue color, serves mainly as the graphical user interface of the framework and is thus meant to ease the interac-

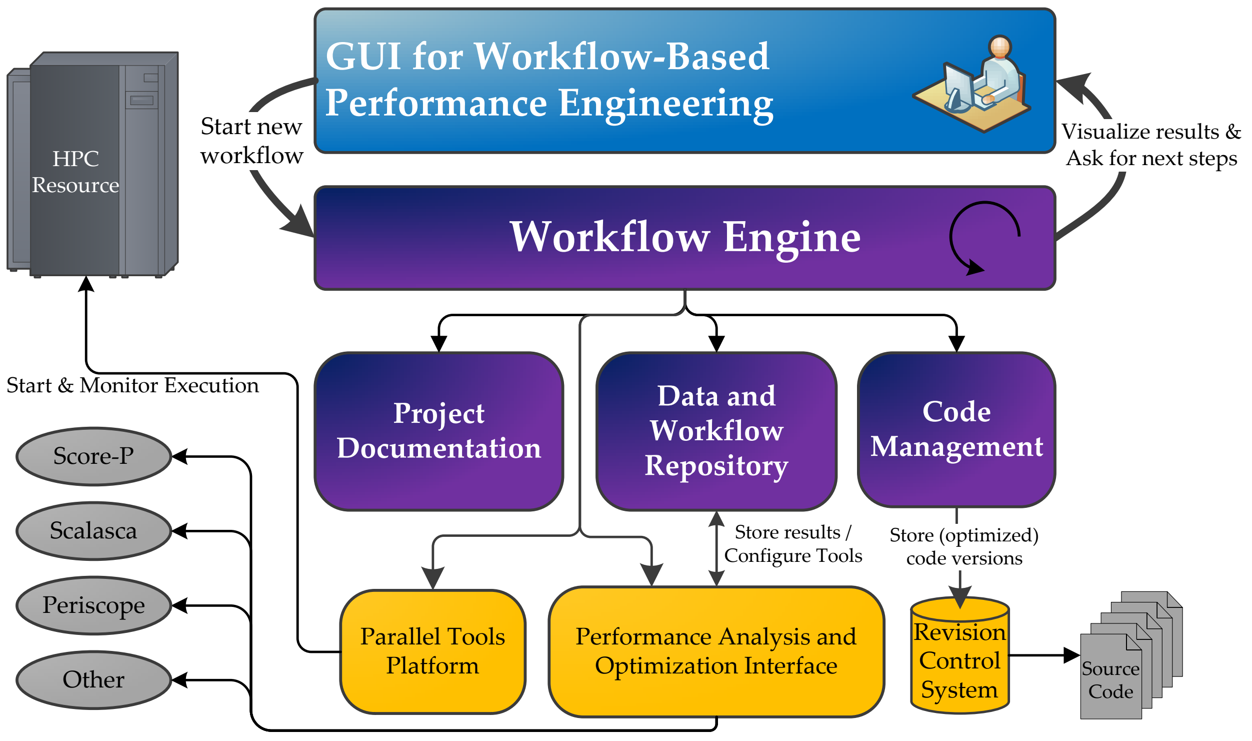

GUI for Workflow-Based Performance Engineering
Workflow Engine
HPC Resource
Start new workflow
Visualize results & Ask for next steps
Start & Monitor Execution
Project Documentation
Data and Workflow Repository
Code Management
Store results / Configure Tools
Store (optimized) code versions
Score-P
Scalasca
Periscope
Other
Parallel Tools Platform
Performance Analysis and Optimization Interface
Revision Control System
Source Code

Figure 7.1.: Overview of the architecture of PAThWay

tion with users. The second layer is represented in the architectural diagram using purple color and provides access to different internal software components that are responsible for converting user-specified high-level workflows into real activities. Examples of such tasks include managing the runtime configuration of experiments and applications, collecting and storing performance results and providing means for documenting the overall performance engineering process. The lowest layer of the framework, shown using orange color, provides interfaces to different external components such as performance tools, a revision control system and the Parallel Tools Platform. The rest of the components in the architectural diagram which are marked using gray color are not directly part of framework. Instead, they can be seamlessly integrated and used within any performance engineering workflows created using PAThWay.

## 7.2. PAThWay Graphical User Interface

The top layer of PAThWay provides the graphical user interface of the system and as such it is a combination of two BPMN editors, custom domain-specific workflow nodes and a set of graphical configuration tools. First and foremost, these enable the user of the framework to execute arbitrary workflows of his application on both local and remote HPC systems. Furthermore, they also provide support for creating new performance engineering workflows by allowing regular users and performance engineers to interactively design new process models and adapt the framework to their own requirements.

The creation of every new performance engineering workflow can start from scratch where the user generates a completely empty model and then has to manually design all specifics of the process by using different BPMN's modeling components and the domain-specific nodes provided by the framework. Alternatively, one of the pre-defined workflow models that come with PAThWay can be imported in the user's project and used as a starting point for the new design. Either way, this phase is supported by the framework using one of the two intuitive Eclipse-based BPMN editors which are provided by the jBPM workflow engine (see Chapter 8.2.2) and the set of custom domain-specific BPMN nodes described in Chapter 8.4.

Once a workflow model is designed, it has to be thoroughly tested. This can be done using either the debug functionality of Eclipse or by activating and monitoring the historical logs provided by jBPM (see Chapter 8.2.1). Finally, after the development and the testing phases of the new model are finished, it can then be directly controlled by using PAThWay's interface for Eclipse. As a result, users will be able to dynamically start the workflow at any time, monitor or temporary pause its execution, visualize performance engineering results and others.

## 7.3. Logic Implementation Layer

The previously discussed top architectural tier depends on many other internal components in order to execute high-level workflows in a platform-independent way. For example, it interfaces with an workflow engine from the layer beneath it to execute the specified processes. Instead of implementing yet another custom workflow execution engine like many other automation projects, for example from the field of Grid computing, the well-established jBPM workflow engine was chosen. It is one of the major open-source workflow engines and as such it has a broad community support and many features that enhance the overall design, execution and auditing of workflows (see Chapter 8.2). As a result of this important design decision, not only a lot of development time was saved for the other components of the framework, but it also tries to bring the architecture of PAThWay to a more standardized and easily portable state.

Additionally, this second layer of the architecture deals with the storage and the management of metadata about any previously performed experiments so that the users can easily monitor the typical performance engineering cycle. As it will be discussed in further details in Chapter 9, the second key architectural decision was to introduce an embedded, Java-based database for the storage of these data. This approach provides a flexible but still structured representation format of the stored information and multiple access means. For example, the data from experiments, i.e. execution configuration, performance results and any user-provided parameters, can be easily queried and processed from other tools in order to provide interactive visualizations or to even create new tuning strategies based on the collected information. Last but not least, this design allows PAThWay to be more easily extended in the future without having to devote significant development time and resources which is fundamental for the success of any complex software system.

As discussed in Chapter 2, having an accessible documentation about the work progress, requirements and ideas is very important for the success of every project. That is why, another major component of the middle layer is designed particularly with the idea of providing a simple documentation environment for experiments and projects. It provides a wiki-based documentation environment where the users can take notes about the performed experiments, refer to existing performance results and other types of project resources such as diagrams, previous experiments or other user-specified notes. As it will be discussed in Chapter 10, this component is internally implemented as an extension of another plug-in for Eclipse called EclipseWiki. This way, different formatting specifications can be supported out-of-the-box together with other features like embedding existing source code and images or linking to external websites and other wiki pages.

An integral part of the performance engineering process is the creation and management of multiple versions of the user's source code. Moreover, one has to switch back and forth between code versions multiple times in order to find the most suitable code modifications. Due to these constant transitions, using a revision control system throughout these activities can be particularly useful. However, in a typical performance engineering process keeping a track of the source code evolution on its own is not always enough. Users must also be able to map the results of any performance experiment to the specific code version used to generate them. This seemingly trivial mapping information is, though, either not recorded by the developers at all or often lost in the large amount of data. This inevitably leads to complications in choosing the right code version to revert to in case of wrong tuning decisions and effectively limits the productivity of performance engineers. In an attempt to tackle this common problem, PAThWay features a specialized code management component within its middle tier. This module, which is further discussed in Chapter 11.2, is capable of autonomously tracking the code evolution and mapping it to the executed performance experiments in a consistent way. Thus, a full end-to-end traceability of the tuning process can be more easily achieved.

## 7.4. Supportive Modules Layer

The lowest layer of the framework contains supportive modules that deal with the low-level management of experiments. One of them wraps around the Parallel Tools Platform introduced in Chapter 5.5.2. This way, PAThWay can easily take advantage of many of PTP's features to provide generic access to and control of different HPC systems. On one hand, for example, when the workflow engine needs to execute a remote process, the framework will internally set up a new network connection using the Remote Tools component from PTP and then start the requested application on that machine. On the other hand, if a workflow requests the execution of a new performance experiment, PAThWay will use a combination of shell scripts and a customized configuration of PTP's JAXB resource manager to interact with the underlying execution system. Based on this setup, the framework is then able to submit and monitor the requested performance experiments in a generic way. Another key advantage that comes with the architectural decision of using the Parallel Tools Platform for an abstraction layer between the workflow engine and the remote HPC systems is the flexibility for adapting PAThWay to new execution configurations. As it will be presented in Chapter 11.4, adding support for a new parallel system involves creating a simple platform-dependent configuration file together with an appropriate entry in the internal database. After these steps, this new machine will become available in any workflow model just like all other predefined ones.

During a typical performance analysis and tuning project, users also need to apply multiple tools in different iterations of the tuning cycle to gain deeper knowledge about the runtime behavior of their codes (see Chapter 1.2). This functionality is implemented in PAThWay at the lowest architectural layer using a custom performance analysis and optimization interface. As it will be discussed in Chapter 9, it uses specialized templates stored in the internal data repository in order to configure the requested performance engineering tools without requiring any modification to the workflow models. These templates specify in a parametrized approach all required commands and arguments, for example, for instrumenting the application, starting profile- or trace-based performance analyses or post-processing the collected results.

In the following chapters, the separate components of all three layers of the PAThWay framework will be described in further details. The specifics about their implementation together with an analysis of how they contribute to addressing the complexity of the performance engineering process in HPC-related projects will be discussed.

Chapter 8

# Workflow Modeling Environment

**Contents**

**8.1. Design Goals** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **86**
**8.2. jBPM Workflow Execution Environment** . . . . . . . . . . . . . . . . . . . . . . . . **86**
8.2.1. Overview and Features . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87
8.2.2. Workflow Model Editors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88
**8.3. Business Process Model and Notation 2.0** . . . . . . . . . . . . . . . . . . . . . . . **90**
8.3.1. Activities . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
8.3.2. Events . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93
8.3.3. Gateways . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96
**8.4. PAThWay's Custom Domain-Specific BPMN Nodes** . . . . . . . . . . . . . . . . **98**
8.4.1. Application Configuration . . . . . . . . . . . . . . . . . . . . . . . . . . . . 101
8.4.2. MPI and OpenMP Configuration . . . . . . . . . . . . . . . . . . . . . . . . 102
8.4.3. Target HPC System Selection . . . . . . . . . . . . . . . . . . . . . . . . . . 103
8.4.4. Performance Tool Selection . . . . . . . . . . . . . . . . . . . . . . . . . . . 104
8.4.5. Experiments Creation and Execution . . . . . . . . . . . . . . . . . . . . . . 105
8.4.6. Runtime Manager Creation . . . . . . . . . . . . . . . . . . . . . . . . . . . 107
8.4.7. Source Code Instrumentation . . . . . . . . . . . . . . . . . . . . . . . . . . 108
8.4.8. Store Additional Information to an Experiment . . . . . . . . . . . . . . . . 109
8.4.9. Load Custom Performance Results to the Database . . . . . . . . . . . . . 110
8.4.10. Execute a Remote Process . . . . . . . . . . . . . . . . . . . . . . . . . . . . 110
8.4.11. Custom BPMN Node for Interactive Questions . . . . . . . . . . . . . . . . 111

## 8.1. Design Goals

There are many different ways to automate iterative processes like the ones found in HPC-related projects. Some people use shell scrips, others build their own custom tools. However, there is no standard approach for the description of such tasks, thus making not only the collaboration among teams harder but also increases the complexity of HPC projects even more.

In order to address these issues, PAThWay integrates a powerful workflow modeling approach based on the broadly accepted industry standard for process modeling called Business Process Model and Notation (BPMN). Furthermore, instead of trying to develop yet another custom workflow engine, the framework is based on the jBPM/Drools suite which allows the creation a very flexible modeling and execution environment.

In the following sections, the underlying workflow engine will be introduced. After that, in Section 8.3, an overview of the most important modeling components of the BPMN standard will be given. Finally, Section 8.4 will describe the usage and the design goals of some of the custom workflow components that are provided with PAThWay in order to adapt the jBPM engine to the domain of performance engineering.

## 8.2. jBPM Workflow Execution Environment

A workflow engine forms the central part of every software system for modeling and automation of processes. As such, the whole architecture of the system and its overall flexibility and acceptance by the future users depends on it. This makes the selection of the proper engine also crucial for the success of PAThWay. As it was discussed in Chapter 3, there are many workflow execution tools each with different advantages and drawbacks. However, the majority of them are commercial software and thus were not applicable for integrating in an open-source framework like PAThWay. After analyzing in detail the available free alternatives, the Java-based jBPM business process management suite from JBoss was finally chosen (see Chapter 3.3.3). This decision was not only motivated by its open design, but also due to the fact that it is currently one of the most stable and feature rich open-source workflow execution engines. As it plays such a central role in the overall architecture of PAThWay, an overview of jBPM's most important features will be given in the following sub-sections together with examples how they are internally used by the framework to create a flexible environment for performance engineering workflows.

### 8.2.1. Overview and Features

One of the key features of jBPM is its complete support for the well-established Business Process Model and Notation 2.0 standard. As discussed previously in Chapter 3.2.4, this specification combines many prior research ideas from the field of process automation in order to create a standardized, tool independent format for modeling and executing processes. The approach of adopting an existing standard versus creating another workflow modeling language as many others did in the past creates a big advantage for PAThWay. For example, the users of the tool can learn the standard much faster as they have access to a broader set of literature and plenty of examples. Moreover, it is much easier to document and share the workflows with other interested research groups in order to improve the collaboration and knowledge exchange within the community. Section 8.3 presents a more detailed description of the supported BPMN 2.0 modeling elements and how they can be used to build performance engineering workflows.

Along the many modeling elements supported by jBPM, there is also support for integrating human activities as part of the workflows. This allows the inclusion of long-running, asynchronous tasks like manual analysis of performance results or implementation of possible tuning modifications. This way, elaborate workflows can be created that support the whole performance engineering cycle discussed in Chapter 1.2. This particular scenario is supported by the engine using the WS-HumanTask standard. Additionally, jBPM provides a generic implementation of a service for managing such human tasks which can be easily adapted to any data and timing requirements imposed by manual activities. Furthermore, a graphical user interface for accessing outstanding manual tasks, providing them with input required by the process or changing their runtime status is included as part of the editors discussed later in this chapter.

As a result of the fact that jBPM is fully implemented in Java, this BPMS is cross-platform allowing one to run the engine in both Windows and Linux without any modifications. Moreover, it can be started as a stand-alone automation server in case of a data-center setup or it can be embedded within an single application to provide a customized execution environment. These features fit very well with the overall architecture of PAThWay as it is designed to be a plug-in for the Eclipse IDE. An extensive management and monitoring application programming interface (API) is also accompanying the jBPM suite. It not only allows to have direct control over the process executions, but also enables the workflow activities to access many other features provided by Eclipse. For example, as discussed in further details in Chapter 11.4, the framework can integrate features of the Parallel Tools Platform into the designed workflows for the sake to providing high-level abstraction and control of miscellaneous HPC systems.

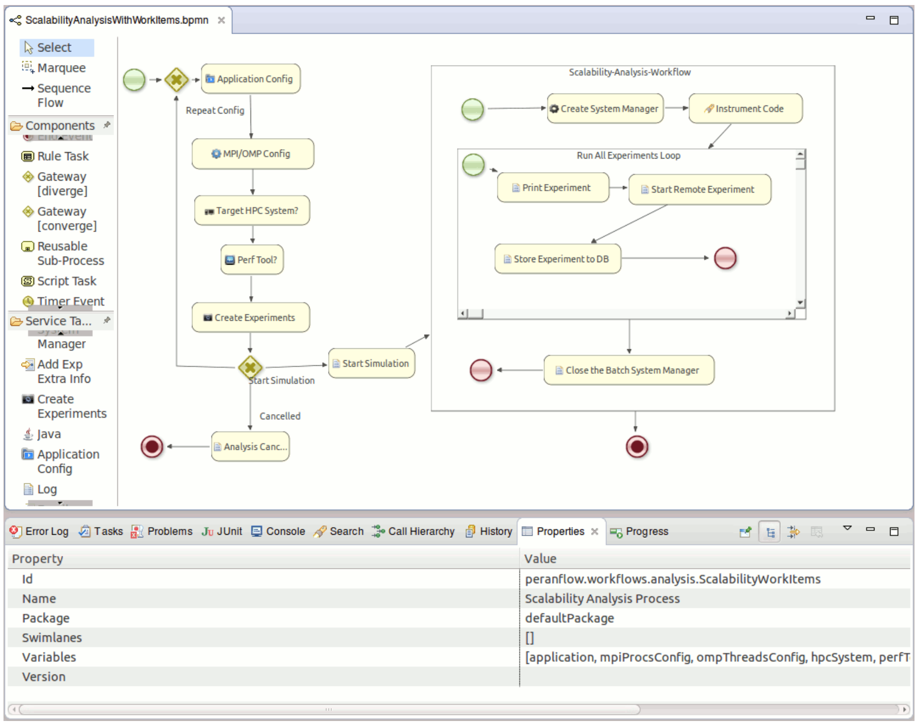
It is very common for performance engineering processes to run over multiple days and even sometimes weeks. For example, the separate steps of a single scalability analysis can stay on the batch queues of some heavily utilized HPC systems for days before they get executed. In addition, these delays tend to be random and generally cannot be forecasted in advance since the operation of shared parallel systems normally depends on multiple factors. In other cases, where workflows including human tasks are used, the completion of the whole process depends on the availability of the involved users and their working speed. As a consequence, the workflow engine must provide a way for temporarily stopping the execution of workflows and be able to resume them from exactly the same point at any future time. Fortunately, this functionality is supported out-of-the-box by the jBPM and can be dynamically activated on a workflow basis [201]. Furthermore, using an external database and the Java Persistence API (JPA), the workflow engine is also capable of persistently storing the complete state of a workflow instance including the values of all involved variables at any point of time during the execution. The saved state can then be restored at any future time and the workflow will automatically continue from the same point as before using exactly the same data.

Last but not least, the jBPM suite provides support for logging the flow of activities in a process together with their timings. This process, also commonly called Business Activity Monitoring (BAM), can store not only the start and end events of the separate tasks of a workflow, but also the value changes of all involved variables. This information can then be analyzed in order to provide valuable insight into the execution behavior of workflows in order to detect potential bottlenecks in the models or create an auditing protocol about the carried out activities.

### 8.2.2. Workflow Model Editors

There are multiple ways users can design new workflow models within the jBPM suite. Depending on the setup of the runtime environment, one can either use a common Web-based user interface or one of the Eclipse-based BPMN editors provided by jBPM's plug-in for Eclipse. Additionally, users can even create new workflows using a plain text editor by following the XML-based BPMN 2.0 standard. This last option is, however, rather complicated and using one of the provided editors is advisable in order to create valid process models.

Currently, there are two different versions of the Eclipse-based editors which are provided by the suite. The original one shown in Figure 8.1 is distributed with the standard installation package but has a bit restricted modeling support. Using it, the available BPMN nodes can be added to a workflow model by simply selecting them from the palette on the left side of the editor and dragging them into the editing canvas on the right side. The *Properties* view of Eclipse can then be used to modify any constrains or possible settings required for the successful execution the nodes. This editor supports most of the BPMN modeling elements and can be easily extended to include additional custom nodes. However, even though jBPM natively supports the newer BPMN 2.0 standard, the original editor cannot be used to design models using this extended version of the specification.



| Property | Value |
|---|---|
| Id | peranflow.workflows.analysis.ScalabilityWorkItems |
| Name | Scalability Analysis Process |
| Package | defaultPackage |
| Swimlanes | [] |
| Variables | [application, mpiProcsConfig, ompThreadsConfig, hpcSystem, perfT |
| Version | |

Figure 8.1.: Original jBPM editor for modeling BPMN workflows using Eclipse

In order to enhance the support for designing models based on the full BPMN 2.0 specification, a new Eclipse editor is being currently developed. Internally, it uses another graphical framework which creates an improved modeling experience with automatic routing of sequence flows and easier alignment of workflow nodes (see Figure 8.2). Moreover, since it includes a much larger set of BPMN elements, it provides selective filtering for the palette on the right side and an enhanced editor for the properties of the nodes. However, at the time of this writing, this newer editor is still under heavy development and unfortunately is not stable for a production environment. As a result, its usage is currently not recommended and thus it is also not included in the official release of PAThWay.

In addition to the two Eclipse-based editors, jBPM features also a Web-based portal. This component targets mainly data-center or enterprise-wide setups and provides a centralized user interface for editing, executing and monitoring of workflows. Furthermore, it is integrated with

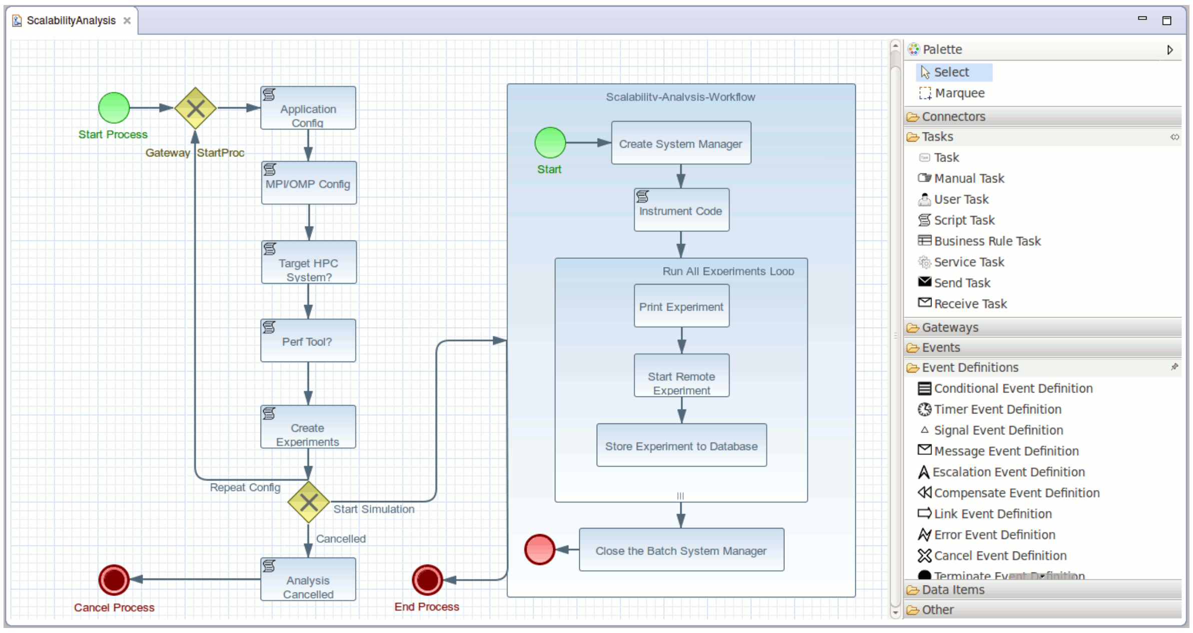


Figure 8.2.: Enhanced jBPM editor for modeling BPMN 2.0 workflows using Eclipse

the Guvnor repository from the Drools project (see Chapter 3.3.3) and thus provides versioning and collaborative design of process models. Last but not least, users can take advantage of the integrated editor for web forms in order to create interactive user interfaces for any of the human tasks included within a workflow model.

## 8.3. Business Process Model and Notation 2.0

Business Process Model and Notation (BPMN) is an established standard for designing and executing process models. As previously discussed in Chapter 3.2.4, it combines both high-level, graphical and XML-based modeling notations and a low-level scheme for the specification of run-time requirements. These key design features make it suitable for a broad spectrum of application areas where a flexible language for structuring and automating iterative processes is required.

PAThWay uses BPMN extensively for providing performance engineering workflows. That is why, some of the most important modeling elements provided by the standard will be discussed in the following sections. Additionally, an overview of domain-specific nodes that were created specifically for the application area of PAThWay will be described. This will be done in order for the user to better understand the framework and learn the foundations needed for designing performance engineering workflows.

### 8.3.1. Activities

Activities are the most fundamental modeling entities included in the BPMN standard. They represent the separate tasks that a workflow has to execute in order to complete the designed process. Activities can be hierarchically structured using multiple levels of sub-processes or they can be stand-alone, top-level nodes that are responsible for a single unit of work. When creating a BPMN workflow using a graphical editor, they are all depicted using a rounded rectangular box with the name of the task in the middle of it. In addition, each node can have an icon specifying its exact type in the upper left corner of the box and an activity marker along the bottom edge (see Figure 8.3).

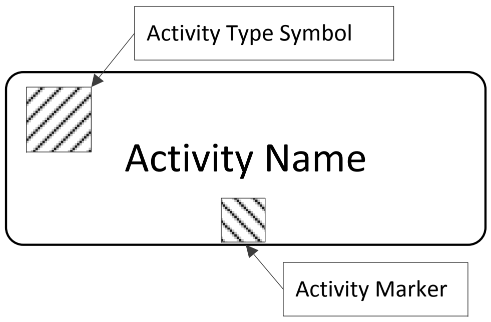


Figure 8.3.: Graphical representation of a BPMN activity

The very simple activities contain neither an additional icon nor a marker. They are typically used for documentation purposes as they do not posses any information about how that task is to be autonomously completed by a workflow execution engine. However, as Table 8.1 shows, the BPMN standard defines other types of activities that provide more control over their runtime characteristics. The most common and flexible types as defined in the standard are [157]:

- Script task - this activity allows the work to be directly specified using a scripting language or another programming language depending on the underlying workflow execution engine. It can be distinguished by having a letter-like symbol in its top left corner.
- Service task - another software-based activity which typically relies on an external Web-service or an automated application for completing its unit of work.
- User task - an activity that describes work performed by human participants using a software application. Its implementation is commonly based on the *Human Task* standard which allows the precise specification of users or groups when distributing the work and can include fine-grained access control. Its graphical notation has a symbol of a human in the top left corner of the rounded rectangle.

- Manual task - similar to the user task above but it is used in cases where the work is explicitly completed manually and without any help from software applications. A hand-like icon is used as its distinguishing symbol.

- Send and receive tasks - using these tasks, the communication between different processes can be designed. They both use messages for the transfer of data and are depicted using a filled envelope symbol for the sending task and an empty one for a receiving side, respectively.

- Business rule task - similarly to jBPM, nowadays most of the workflow engines are part of bigger software systems that feature additional rule execution engines. This way, many workflow activities, e.g. for data analysis, can be modeled more efficiently using high-level rules instead of being implemented using low-level programming languages. In a graphical model, these tasks can be identified by the symbol of a table in the top left corner.

- Domain-specific task - quite often certain workflow activities are so complex that they cannot fit into a script task nor can be implemented using generic rules. Even though a service task can be used in this situation, it still relies on the existence of an external Web-service or an application which limits its applicability. Consequently, many workflow engines include support for an additional type of activity which allows custom, domain-specific work units to be encapsulated as new BPMN nodes. These can then be reused in any workflow model as if they were natively supported by the standard. The graphical notation of the domain-specific tasks can include any custom icon and as such are not included in Table 8.1.

As mentioned previously, BPMN activities can be stand-alone or can include other tasks. Furthermore, they can be either designed to run one single time, in a loop or on all elements in a given data collection. That is why, the official standard separates activities further in the following groups:

- Sub-Process - this task can either be used as a container for other activities in order to achieve logical separation of the tasks in a workflow or can be embedded into other existing workflow models. At any time, a sub-process can be expanded showing all child activities or collapsed. An activity being in the later state can be distinguished by having a square with a plus sign in the bottom center part of its graphical notation.

- Loop - this activity is also a sub-process which can be used to model a simple *while*-loop. It will repeat the tasks contained in it as long as its logic condition evaluates to `true`.

Table 8.1.: An overview of activity types and markers supported by BPMN 2.0 [157]

| Activity Types | Activity Markers |
|---|---|
| 📜 Script Task | ⊞ Sub-Process |
| ⚙ Service Task | ⟲ Loop |
| 👤 User Task | ⦀ Parallel Multiple Instance |
| ☞ Manual Task | ≡ Sequential Multiple Instance |
| ✉ Send Task | ~ Ad-Hoc Process |
| ✉ Receive Task | ⏪ Compensation |
| ▦ Business Rule Task | |

- Parallel Multiple Instance - this is another task similar to a sub-process which defines a *foreach*-loop. It runs the activities it contains on all entries of a given collection in parallel. It is depicted using three vertical lines as in the bottom center part of the rounded rectangle and can additionally include a square with a plus inside it in case it is in a collapsed state.

- Sequential Multiple Instance - similar to the previous task but runs the child activities in a sequential manner. This task is also depicted with three lines but they are horizontally position instead of vertically.

- Ad-Hoc Process - a type of a sub-process where all child activities are specified without giving information about their sequence flow relations. Once an ad-hoc process starts, the included tasks decide on their own if they have to run or not. The graphical notation for the ad-hoc activity is a tilde symbol.

- Compensation - a sub-process that is designed to handle a special compensation event which is discussed later on in this section.

### 8.3.2. Events

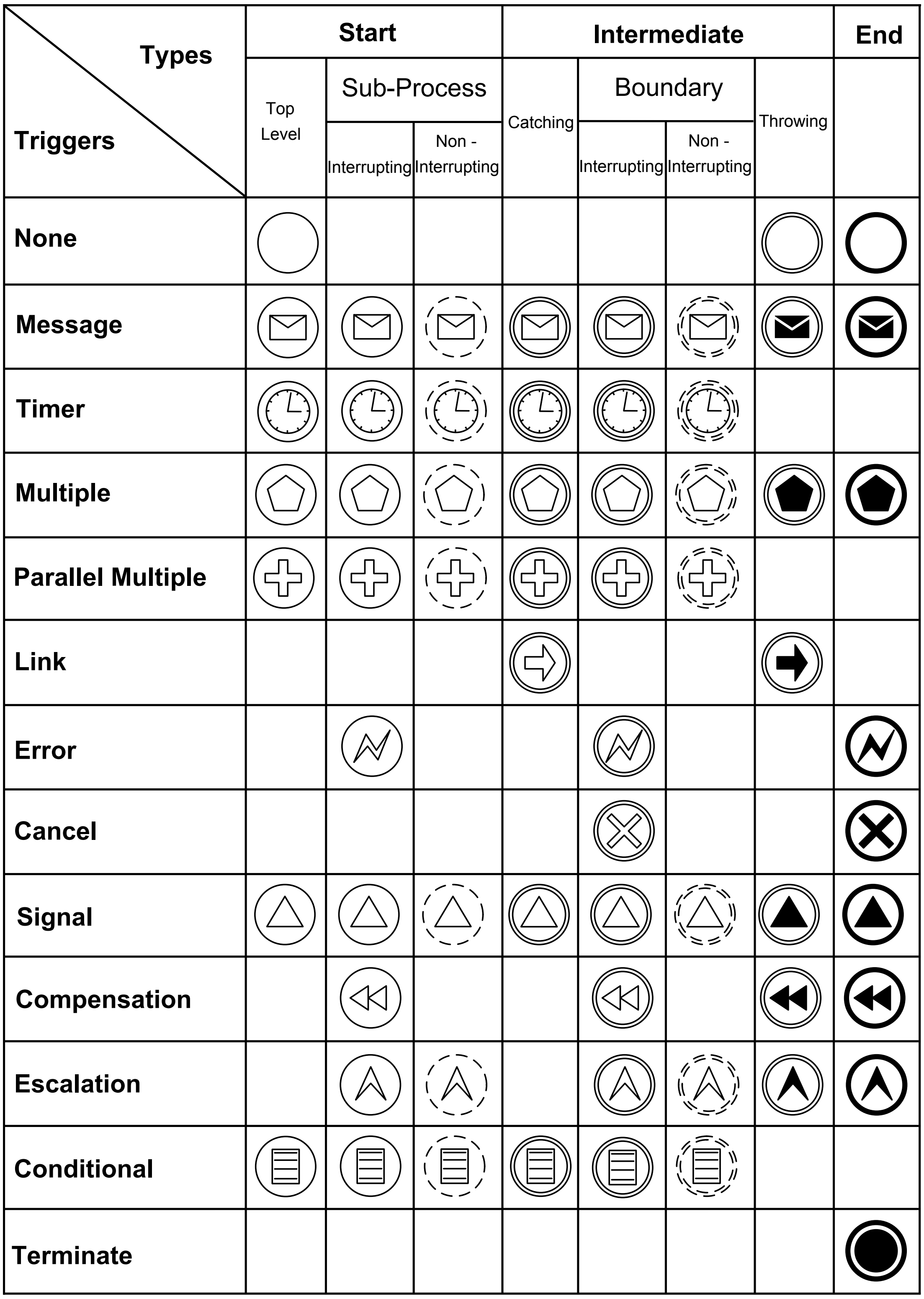
Events represent the reactive components of a workflow which can respond to external and internal state changes. As it can be seen in Table 8.2, the events in BPMN can be separated in three major types based on how they interfere with the process they are part of:

- Start events trigger the execution of a top-level process or a sub-process, respectively. The second group can be further split into events that interrupt the execution of the parent process and those that do not do that. Start events are always depicted with a circle having a single thin line.
- Intermediate events occur while a process is running. They can be separated in two groups: catching events that can be either positioned inside of a process or on the boundary of a sub-process and are able to react to other events; and throwing ones that can generate new events during the runtime of a process. The graphical notation for all intermediate events is a circle having two thin lines.
- End events signal the termination of the whole process or just a sub-process of it. Optionally, they can also send messages or generate exceptions, signals or other events. They are graphically represented using a circle with single thick line.

Moreover, since events are the reactive elements in a workflow, the BPMN standard [157] splits them further in groups depending on the type of state change that is allowed to trigger them:

- None - represents a generic event type without any specific event trigger. This is, for example, the manual start of a process by the user or the automatic start of a sub-process by the parent once it is reached.
- Message - these events are triggered by the receipt of a message from another process or a different pool of the current process.
- Timer - an internal timer that will generate an event on a predefined date and time or after a certain amount of time has elapsed, for example, like a time-out event.
- Multiple - a generic trigger that can be activated by multiple sources (e.g. messages, timers, etc.) where only one of them suffices for starting the process attached to the event.
- Parallel Multiple - similar to the multiple trigger above but requires that all defined event sources get activated before starting the consecutive process.
- Link - event trigger type that is used to change the sequence flow of a process by connecting different process parts together. It is similar to a *GoTo* statement available in other programming languages.
- Error - triggers like this one generate runtime exceptions in order to signal for the erroneous state of a workflow and normally abort the execution of the current process. These generated exceptions can then be caught by other error event types positioned on process's boundary.

Table 8.2.: An overview of all events supported by BPMN 2.0 [157]



| Types / Triggers | Start | | | Intermediate | | | | End |
|---|---|---|---|---|---|---|---|---|
| | Top Level | Sub-Process | | Catching | Boundary | | Throwing | |
| | | Interrupting | Non - Interrupting | | Interrupting | Non - Interrupting | | |
| **None** | ○ | | | | | | ○ | ○ |
| **Message** | ✉ | ✉ | ✉ | ✉ | ✉ | ✉ | ✉ | ✉ |
| **Timer** | 🕒 | 🕒 | 🕒 | 🕒 | 🕒 | 🕒 | | |
| **Multiple** | ⬠ | ⬠ | ⬠ | ⬠ | ⬠ | ⬠ | ⬟ | ⬟ |
| **Parallel Multiple** | ✚ | ✚ | ✚ | ✚ | ✚ | ✚ | | |
| **Link** | | | | ⇨ | | | ➡ | |
| **Error** | | ϟ | | | ϟ | | | ϟ |
| **Cancel** | | | | | ⊗ | | | ⊗ |
| **Signal** | △ | △ | △ | △ | △ | △ | ▲ | ▲ |
| **Compensation** | | ⏪ | | | ⏪ | | ⏪ | ⏪ |
| **Escalation** | | ⋀ | ⋀ | | ⋀ | ⋀ | ⋀ | ⋀ |
| **Conditional** | ≣ | ≣ | ≣ | ≣ | ≣ | ≣ | | |
| **Terminate** | | | | | | | | ● |

- Cancel - a very specialized event trigger that cancels the current process's transaction in a transaction-based workflow model.
- Signal - similar to the message-based event triggers but without specifying a destination. Instead, signals are broadcast to all process levels and can be caught in any interested parts of a workflow using a catching signal event type.
- Compensation - a trigger that can get activated on the completion of an activity in order to perform additional finalization tasks.
- Escalation - similar to the error event triggers but do not abort the process's execution.
- Conditional - these event types get triggered when a predefined expression evaluates to `true`.
- Terminate - depicts a simple event that terminates the current process and all of its parent processes.

### 8.3.3. Gateways

The last group of BPMN components which are fundamental for understanding existing and designing new workflows contains the so called gateways. These nodes are responsible to controlling and deviating the sequence flow of a process model based on predefined logical conditions. Moreover, they can be used to split the flow of the process in branches that can be processed concurrently. That is why, these gateways are also commonly called split and join nodes in order to describe their inherent usage. Characteristic for all split gateways is that they always have a single incoming sequence flow and two or more outgoing ones. Opposite to them, the join gateways can have more than two incoming connections but must posses only a single outgoing sequence flow.

As shown in Table 8.3, there are two major types of gateways: those that take decisions based on existing data and the ones that control the flow based on events. The first type of nodes can represent both split and join nodes depending on their position in the workflow. However, the event-based gateway can only serve as split nodes where the joining behavior is normally modeled using data-based gateways.

When designing BPMN workflows, the following data-base gateways can be used:

- Exclusive OR (XOR) gateway - when used as a split gateway, this node consecutively evaluates all logical conditions which are bound to its outgoing ports according to their priority. Then, it transfers the sequence flow to only the first one that evaluates to `true`. In case

Table 8.3.: An overview of the gateway nodes supported by BPMN 2.0 [157]

| Data-Based | Event-Based |
|---|---|
| Exclusive OR | Exclusive, Start OR |
| Inclusive OR | Exclusive, Intermediate OR |
| And / Parallel | Parallel |
| Complex | |

none of the predefined conditions are fulfilled, the workflow takes the default path initially specified by the user.

- Inclusive OR (OR) gateway - in contrast to the exclusive gateway, this node activates all outgoing sequence flows whose logical conditions are fulfilled instead of just the first one.
- Parallel gateway (AND) - once the sequence flow reaches this type of a gateway, it gets split in multiple concurrent branches without having to comply with any logical conditions. Once the work in these branches is completed, they should be merged back together using another parallel gateway.
- Complex gateway - this node allows the specification of logical expressions where multiple outgoing sequence flows can be executed at anytime. In contrast to the parallel gateway, it does not activate all outgoing ports simultaneously but rather in a group-wise fashion depending on their logical conditions.

As already mentioned, the second approach of implementing logical branching in BPMN-based workflows is based on events. Currently, the standard specifies three types of event-based gateways (see Table 8.3). The first two are both exclusive nodes which means that they will activate only one outgoing branch depending on the events bound to them and the order in which they arrive. The only difference between them is in their usage: the first gateway is used to start a sub-process and as such does not contain any incoming flows whereas the second one simply deviates the intermediate sequence flow of a process. The last type of an event-based gateway is the parallel one. It is similar to the exclusive start node but can get triggered multiple times on the arrival of each of its predefined events.

## 8.4. PAThWay's Custom Domain-Specific BPMN Nodes

As already discussed, every workflow model is made of a set of control and dataflow nodes including events, stand-alone tasks and sub-processes, conditional gateways and others. At the same time, most of the workflow engines also support custom domain-specific nodes which provide means for codifying the logic of activities in order to allow for more flexible workflow designs. However, since the standard does not explicitly specify these nodes, the support for them is heavily dependent on the underlying workflow execution engine. Some BPMN tools do not include support for domain-specific nodes at all, others provide ones that are only based on simple semantic definitions or Web services and there are also other tools that include components which can be implemented using general programming languages such as C++ or Java.

An example from the last group is the jBPM workflow engine that is used internally in the framework. It allows developers to create custom domain-specific nodes, internally called work items in the jBPM terminology, to encapsulate complex activities using the Java programming language. There are, however, specific requirements that must be fulfilled in order for the users to be able to reuse these nodes in their workflow models.

Firstly, a high-level definition of the node has to be created so that the workflow editors can interpret its runtime requirements and provide the user with the ability to set the parameters as part of the process model. Such a definition has to be created using the MVEL [29] expression language format and must at least include the *name* of the custom BPMN node, its input *parameters* and the output *results*. As it can be seen in Listing 8.1, the names of all variables are specified in textual form and their respective data types have to be explicitly defined. These steps are required for jBPM to correctly interpret and check user's input for correctness.

The workflow engine provides a multitude of data types, e.g. strings, intergers, lists, that cover the majority of use-cases. However, when a custom class has to be used as the data type of a parameter, it can be easily wrapped in an *ObjectDataType* parameter as done for the *Experiments* in the previous code listing. Other custom data types can also be created and used in the definition by implementing a special interface. However, this will not be discussed here as it is a very rare use-case.

Additionally, three more fields can be defined for each domain-specific node in order to customize its graphical representation. Using *displayName* the externally visible name of the node can be changed. An icon can also be attached to the graphical notation using the *icon* field respectively. Last but not least, a custom GUI for setting the values for the parameters at design time can be specified using the *customEditor* attribute.

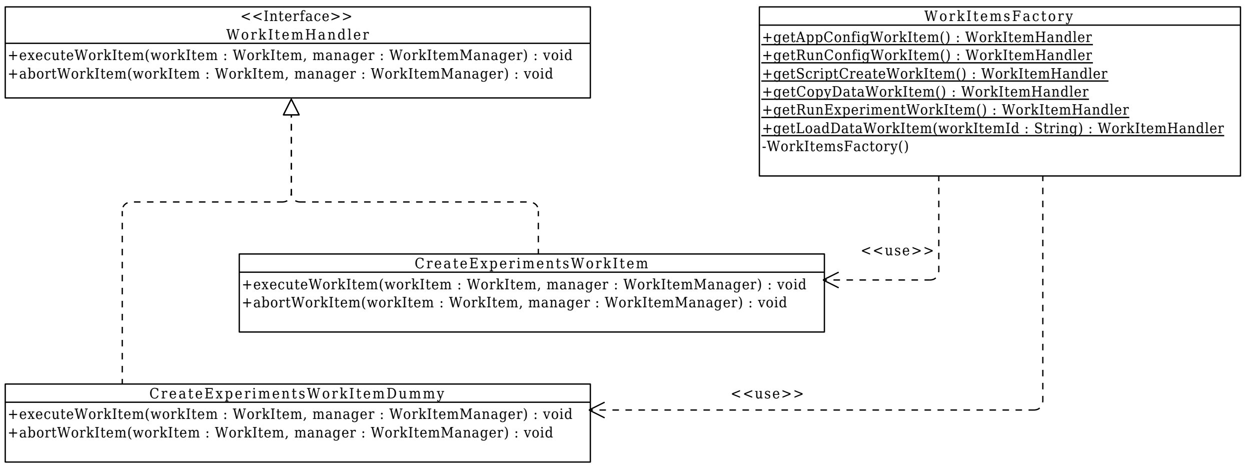
Listing 8.1: Excerpt from the PAThWayWorkDefinitions.conf file defining all PAThWay's domain-specific BPMN nodes

```
import org.drools.process.core.datatype.impl.type.ObjectDataType;
import org.drools.process.core.datatype.impl.type.StringDataType;
import pathway.data.persistence.Experiment;

[
    "name" : "CreateExperiments",
    "parameters" : [
                "Application" : new ObjectDataType(),
                "MPIProcs" : new StringDataType(),
                "OMPThreads" : new StringDataType(),
                "HPCSystem" : new ObjectDataType(),
                "PerfTool" : new ObjectDataType()
    ],
    "results" : [
        "Experiments" : new ListDataType(new ObjectDataType("Experiment")),
        "ConfigOK" : new StringDataType()
    ],
    "customEditor" : "pathway.workflows.editors.CreateExperimentsEditor",
    "displayName" : "Create Experiments",
    "icon" : "icons/system_window.png"
]
```

Secondly, this newly created node definition has to be registered with the jBPM engine for it to become available in the workflow editors. A common approach is to create a single file with definitions of all custom BPMN work items and add a link to it in the configuration of the workflow engine. In this manner, PAThWay also registers all its custom workflow nodes by specifying the file containing the definitions (*PAThWayWorkDefinitions.conf*) in the static jBPM configuration file:

```
drools.rulebase.conf:
      drools.workDefinitions = PAThWayWorkDefinitions.conf
```

Additionally, in order to activate this customized design and runtime configuration in jBPM, both *drools.rulebase.conf* and *PAThWayWorkDefinitions.conf* files should be explicitly added to the class path of the workflow engine. Depending on the execution mode of PAThWay, i.e. standalone or as part of Eclipse, there are different approaches of doing so. In the first case, the path to the files can be directly specified as a command-line argument during start-up. In the second case, the Eclipse project containing user's workflow models should be converted first to a Java project and then its class path has to be extended to include PAThWay's library container. These steps can also be followed to integrate other user-defined domain-specific BPMN nodes at any later stage.



<<Interface>>
WorkItemHandler
+executeWorkItem(workItem : WorkItem, manager : WorkItemManager) : void
+abortWorkItem(workItem : WorkItem, manager : WorkItemManager) : void
WorkItemsFactory
+getAppConfigWorkItem() : WorkItemHandler
+getRunConfigWorkItem() : WorkItemHandler
+getScriptCreateWorkItem() : WorkItemHandler
+getCopyDataWorkItem() : WorkItemHandler
+getRunExperimentWorkItem() : WorkItemHandler
+getLoadDataWorkItem(workItemId : String) : WorkItemHandler
-WorkItemsFactory()
CreateExperimentsWorkItem
+executeWorkItem(workItem : WorkItem, manager : WorkItemManager) : void
+abortWorkItem(workItem : WorkItem, manager : WorkItemManager) : void
<<use>>
CreateExperimentsWorkItemDummy
+executeWorkItem(workItem : WorkItem, manager : WorkItemManager) : void
+abortWorkItem(workItem : WorkItem, manager : WorkItemManager) : void
<<use>>

Figure 8.4.: Implementation of custom BPMN nodes using the jBPM workflow engine

Thirdly, a separate Java-based runtime handler has to be developed for each of the previously defined nodes for representing their logic. As it can be seen from UML diagram in Figure 8.4, each handler has to implement the *org.drools.runtime.process.WorkItemHandler* interface in order to enable its autonomous runtime execution within a workflow. The normal behavior of a node can then be specified in the *executeWorkItem* function. In case of a runtime failure or explicit cancellation of the execution by the user, the jBPM workflow engine will always call the *abortWorkItem* function.

Finally, the developed node handlers have to be mapped to their definitions, respectively. This can be accomplished dynamically at runtime using either PAThWay's *WorkItemsFactory* factory class or directly instantiating the handler classes. Alternatively, they can also be statically mapped by providing a configuration file like the one shown below in Listings 8.2.

Listing 8.2: Excerpt from the PAThWayWorkItemHandlers.conf file mapping previously defined domain-specific BPMN nodes to their runtime handlers

```
import pathway.workflows.workitems.*;
[
    "AppConfig" : new AppConfigWorkItem(),
    "ProcThreadsConfig" : new ProcThreadsConfigWorkItem(),
    "SelectTargetSystem" : new SelectTargetSystemWorkItem(),
    "SelectPerformanceTool" : new SelectPerformanceToolWorkItem(),
    "StartRemoteProcess" : new StartRemoteProcessWorkItem(),
    "LoadDataPeriscope" : new LoadDataPeriscopeWorkItem(),
    "CreateExperiments" : new CreateExperimentsWorkItem(),
    "RunExperiment" : new RunExperimentWorkItem()
]
```

Similarly to the second step, all configuration files containing mappings of runtime handlers to domain-specific node definitions have to be added to the class path of jBPM. However, due to the fact that these handlers are only available at runtime and thus are not relevant for any editors, they have to be specified in the session configuration file of the engine:

```
drools.session.conf:
        drools.workItemHandlers = PAThWayWorkItemHandlers.conf
```

As briefly discussed in previous chapters, PAThWay provides a collection of domain-specific nodes. They serve not only as an interactive user interface during the workflow's execution but also enable the seamless integration of the underlying software components of the framework within the workflow models. This way, many of the requirements posed by the typically inhomogeneous nature of performance analysis and tuning projects can be addressed in order to create a convenient environment for designing and executing performance engineering workflows. In the following subsections, the domain-specific BPMN nodes provided by PAThWay will be presented in more detail and their target application will be discussed.

### 8.4.1. Application Configuration

One of the very first things users have to choose when starting a new performance engineering experiment is the application they would like to analyze. However, simulation codes typically have multiple input sets and plenty of environmental and command-line arguments. Moreover, loading additional library modules may also be required for a successful execution.

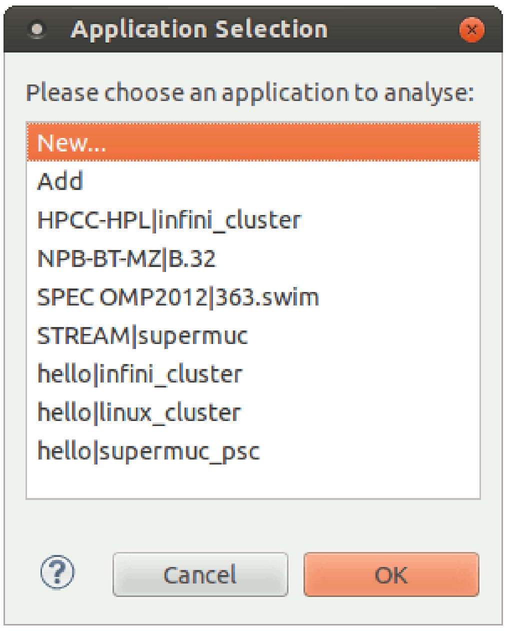
In order to enable a convenient selection of these settings, PAThWay stores internally the location of applications together with their runtime configurations (see Chapter 9.2.2) and provides a custom BPMN node for accessing them from any workflow model. This component interactively asks the user to choose which one of the predefined applications should be consecutively used within the workflow (see Figure 8.5b). If none of the available ones fits with the execution requirements, the user will then be given the option to create a new configuration entry. After the node completes, it will return a tuple with the name and the configuration of the selected application which can then be used during the creation of experiments.

Similarly to any other BPMN component, there are two different approaches for defining the *Application Selection* node in a workflow: using one of the provided editors and node's graphical notation shown in Figure 8.5a or directly configuring it with the help of the XML notation provided by BPMN as can be seen in Appendix B.1.

(a) Graphical notation

Application Config

(b) Runtime user interface



Please choose an application to analyse:

New...
Add
HPCC-HPL|infini_cluster
NPB-BT-MZ|B.32
SPEC OMP2012|363.swim
STREAM|supermuc
hello|infini_cluster
hello|linux_cluster
hello|supermuc_psc

Figure 8.5.: Custom BPMN node for the selection of an application to be analyzed

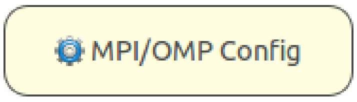
### 8.4.2. MPI and OpenMP Configuration

Typically, right after the selection of the application to be used within a performance engineering workflow, the proper runtime environment has to be configured. Due to the nature of PAThWay's target field for automation and the standard libraries involved in the parallel execution of HPC codes, this step includes the specification of the number of MPI processes and OpenMP threads.

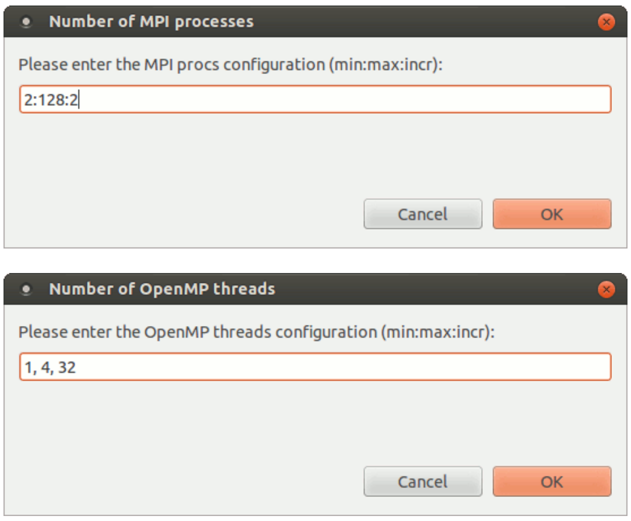
Providing the number of processes and threads is supported in PAThWay by another domain-specific BPMN node. Using the two dialog boxes shown in Figure 8.6b, it gives the option to enter either a single number for the amount of threads or processors, a comma-separated list of values or a whole range of values in the format `start:end:increment multiplier`. Upon completion of this node, these entries should be propagated to the experiments creator node. There they will be expanded accordingly and used to configure all new performance engineering experiments.

This domain-specific node does not require any input parameters and returns the number of both MPI processes and OpenMP threads. In a performance engineering workflow, it can be integrated by the users using the graphical representation shown in Figure 8.6a or the XML-based definition from Appendix B.2. In both cases, however, the results of the BPMN node have to be explicitly mapped to existing variables so that they can be reused by other consecutive components in the workflow model.

(a) Graphical notation



(b) Runtime user interface



Please enter the MPI procs configuration (min:max:incr):

2:128:2

Please enter the OpenMP threads configuration (min:max:incr):

1, 4, 32

Figure 8.6.: Custom BPMN node for the configuration of the number of MPI processes and OpenMP threads

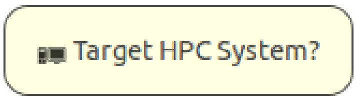
### 8.4.3. Target HPC System Selection

After choosing an application and configuring any required runtime settings, performance engineers normally have to decide which HPC system they are going to use for the new experiments. Often, multiple choices are available each having its specific access requirements, different underlying hardware and software infrastructures and various execution restrictions. For example, some HPC sites allow users to log in for a predefined set of machines and using only public-key authentication. Moreover, some systems are frequently configured to refuse serial or even small-scale parallel jobs. As a result, users have to commonly use many different HPC systems throughout their work and providing an easy way of choosing the right one during the execution of performance engineering workflows is crucial for improving their productivity.

That is why, the framework has the ability to store many details about different HPC systems in its internal data repository (see Chapter 9.2.3). In order to provide access to these data from any workflow model, it also implements a domain-specific node that upon execution will provide the user with a list of all currently preconfigured machines similar to that from the screenshot shown in Figure 8.7b.

Similarly to the previous two custom nodes, this one can also be added to a workflow model by either inserting the definition from Appendix B.3 or using its graphical representation shown in Figure 8.7a which can be selected from the palette of one of jBPM's workflow editors.

(a) Graphical notation



(b) Runtime user interface

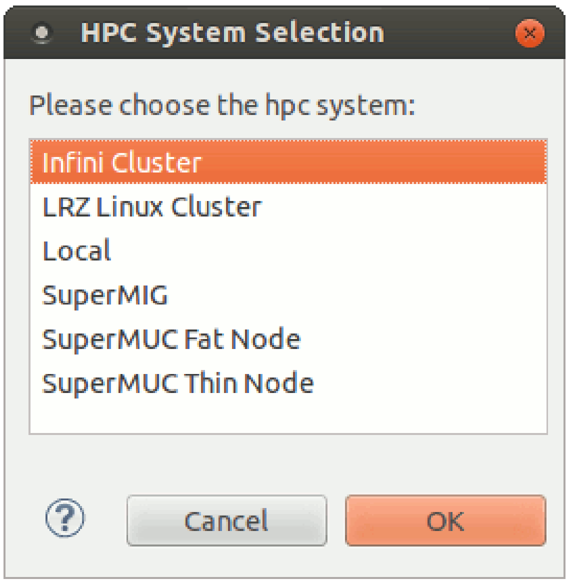


Figure 8.7.: Custom BPMN node for the selection of a target HPC system

### 8.4.4. Performance Tool Selection

As previously discussed, PAThWay is designed to serve as a performance engineering automation environment that aims to integrate different high-performance analysis and tuning tools. Being developed in the framework of the Periscope project, it can currently work out-of-the-box with the measurement library Score-P and the performance analysis tools Scalasca and Periscope. However, it can be easily extended to integrate other tools as well.

Similarly to the way PAThWay handles preconfigured HPC systems and user applications, the framework saves runtime settings and meta information about the separate performance analysis tools in its database. As it will be presented in Chapter 9.2.4, this information includes parametrized start-up commands, supported analysis modes and others. Consistently, data about the code instrumentation support of each tool is internally stored so that the framework can instrument on-the-fly the source code chosen by the user (see Section 8.4.7). This way, different analyses can be easily executed as part of a typical performance engineering cycle.

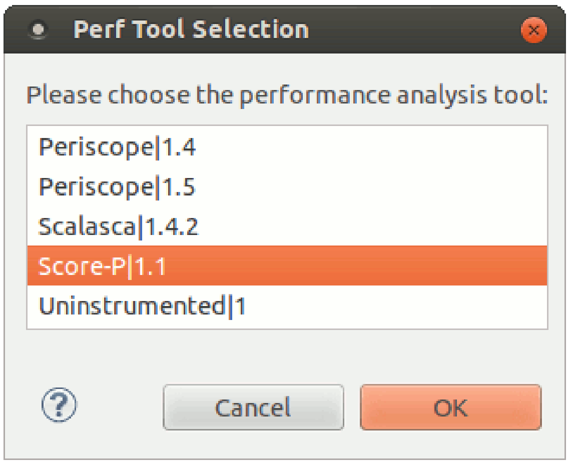
Like the previous domain-specific nodes, PAThWay also provides a custom component for the dynamic selection of a target HPC system. Its graphical notation, presented in Figure 8.8a, is accompanied by a detailed XML-based definition available in Appendix B.4.

At runtime, this BPMN node will query PAThWay's internal database, generate a list of all pre-existing machines and interactively ask the user to choose one using a UI similar to the one from

Figure 8.8b. After that, it will return the tag of the selected performance tool together with its version so that these can be used by the experiments creator to configure the requested performance engineering studies.

(a) Graphical notation

(b) Runtime user interface



Please choose the performance analysis tool:

Periscope|1.4
Periscope|1.5
Scalasca|1.4.2
Score-P|1.1
Uninstrumented|1

Figure 8.8.: Custom BPMN node for the selection of a performance tool

### 8.4.5. Experiments Creation and Execution

Whenever the interactive runtime configuration is finished, the next step in a typical workflow creates performance experiments based on that previously provided information. This is achieved using the *Create Experiments* domain-specific BPMN node shown in Figure 8.9a. Its design goal is to expand the provided MPI and OpenMP configurations, query the internal database for the complete configurations of both the user's application and the chosen performance tools and prepare other execution details specific to the target HPC system.

Firstly, the framework generates a single experiment for each unique combination of a number of MPI processes and OpenMP threads. For example, if the user has specified **(2:32:4)** for the MPI processes and **(4, 8, 32)** for the number of OpenMP threads, the framework generates and automatically configures the following nine combinations:

(# MPI processes, # OpenMP threads):

| | | |
|---|---|---|
| ( 2, 4) | ( 2, 8) | ( 2, 32) |
| ( 8, 4) | ( 8, 8) | ( 8, 32) |
| (32, 4) | (32, 8) | (32, 32) |

Secondly, the internally stored configurations of the selected performance tool and the chosen application are retrieved and packed together with each of the previously generated experiments. This way, each experiment contains not only the number of MPI processes and OpenMP threads but also information about all command-line arguments, filesystem paths to both the source code and the location where the compiled executable is stored on the target HPC machine and any required environment variables and external modules. In case multiple performance tools or application configurations are requested by the user, a new copy of each of the experiments will be created and configured for each of the selected combinations of performance tools and applications, respectively.

Thirdly, the data about the target HPC system will be fetched from the database and its consistency will be checked. This includes, for example, a check whether the required remote machine private-key exists or if an SSH tunnel is needed and has already been created by the user. Once these checks pass, all the required login information is also packed together with the experiments just like previously done for the configurations of the performance tool and the application. As a result, a set of self-contained experiment definitions will be created. This can then be either further extended and processed or simply executed depending on the inherent logic of the given workflow.

Lastly, the framework will show a summary about the generated experiments which looks similarly to the screenshot in Figure 8.9c and the user will be given the chance to either confirm them, ask for the interactive configuration process to be repeated or immediately quit the workflow without any further processing.

After the complete list of experiments is successfully created, it can then be processed either in parallel or sequentially using standard BPMN constructs like loops or multiple instance BPMN activities. Among others, however, these nodes should always contain another of PAThWay's domain-specific components: the *Run Experiment*. It is graphically represented with the rounded box shown in Figure 8.9b and as an input this custom node takes a batch system manager and an experiment object (see the XML definition found in Appendix B.7). Its main purpose is to connect the high-level workflow components with the underlying runtime execution and monitoring system in order to start of a new performance engineering experiment.

It has to be mentioned, however, that this node is designed as an asynchronous one, i.e. it will not block and wait until the started experiment completes. Instead, it will return immediately so that the workflow engine can continue with the execution of the next activities in the process model. This behavior is motivated by the fact that typical HPC jobs often run for hours which would have otherwise led to blocking the whole workflow engine and eventually making it unstable. Moreover, since the jBPM is capable of persisting the state of a running workflow only

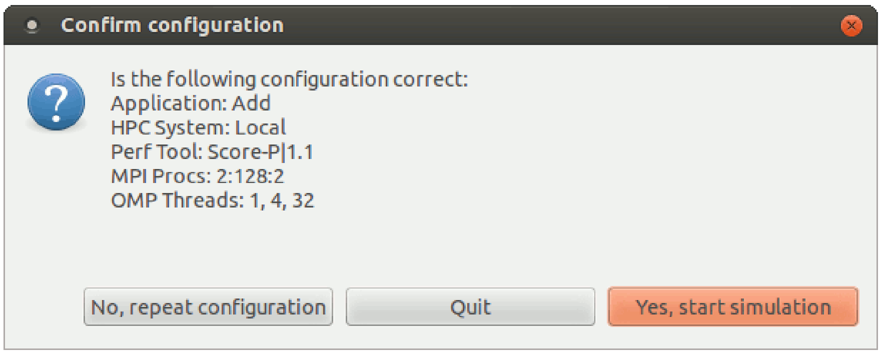
(a) BPMN node for creating performance experiments based on the provided by the user parameters

Create Experiments

(b) BPMN node for executing a preconfigured experiment

Run Experiment

(c) Confirmation of the selected runtime configuration


Confirm configuration

Is the following configuration correct:
Application: Add
HPC System: Local
Perf Tool: Score-P|1.1
MPI Procs: 2:128:2
OMP Threads: 1, 4, 32

Figure 8.9.: Custom BPMN nodes for the creation and execution of performance engineering experiments

when no node is currently being executed, pausing or restarting of processes is not possible in the case of synchronous node behavior.

### 8.4.6. Runtime Manager Creation

The runtime manager of PAThWay serves as the main experiments orchestrator behind all the workflows. It provides an interface between the *Experiments Execution* node and the internal execution and monitoring system. Its design goal is to manage all local and remote experiments requested by a given performance engineering study by interacting with underlying software components of PAThWay like the code versioning module, the PTP wrapper for abstracting of HPC systems and the component for detection of the execution environment. Furthermore, it is also responsible for maintaining network connections with the target HPC machine and collecting the results from managed experiments once they become available. Last but not least, the runtime manager is also in charge of updating the internal database with runtime details about the experiments such as execution status, location of results and others.

Even though some workflows might be designed to interactively collect data from the users or operate only on local resources, most of the time the performance engineering process requires interaction with external resources. In this case, every workflow model must create a separate

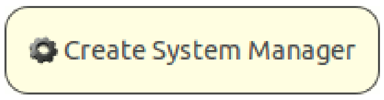


Figure 8.10.: Custom BPMN node for the creation of a new runtime manager

runtime manager for each of the accessed HPC system. This activity is supported by the *Create System Manager* domain-specific node that is provided with PAThWay. It is depicted as the rounded box shown in Figure 8.10 and requires three parameters as it can be seen from its XML definition in Appendix B.8. However, from all of them only the *Experiment* parameter is mandatory as it is used to provide all the necessary details about the target system and the requested execution environment.

Once this node completes, it will return a special batch system manager object. Similarly to the interactive configuration node discussed previously, it also has to be mapped to an internal variable in the workflow model in order to be successfully propagated to any of the consecutive BPMN nodes. For example, the *Experiments Execution* workflow activity requires an existing batch system manager for it to be able to submit a new performance engineering experiment and will, respectively, generate an error if none is specified.

### 8.4.7. Source Code Instrumentation

Performance engineering includes many different activities. However, the instrumentation of source code is almost always part of it as it is an important precondition for monitoring the runtime execution of applications. As it was discussed previously in Chapter 5, this task can include different types of instrumentation such as source-to-source, binary, compiler-based and others. However, the choice of which particular technique should applied and how it is realized depends only on the groups developing performance analysis tools. As a result, the majority of them have their own instrumentation tools that have completely different command-line arguments and require diverse external libraries.

In order to enhance this activity, the configuration of each performance tool, stored in PAThWay's internal database, contains additional parametrized entries about tool's specific instrumentation support. Based on this information, the choice of which tool should be used in any particular experiment can be postponed to runtime, thus allowing the design of more generic workflows. Furthermore, complex processes requiring multiple re-instrumentation steps can also be easily created. For example, a typical use-case is doing a profile run in order to gather the high-level runtime statistics, then deciding whether a deeper tracing analysis is needed and then re-instrumenting the code and starting a new experiment if required.

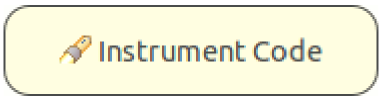


Figure 8.11.: Custom BPMN node for instrumenting user's code as part of a performance engineering experiment

Because the source code instrumentation plays such an important role in the standard performance engineering process, PAThWay supplies another domain-specific node that allows for the easy integration of this activity as part of any workflow model. It is graphically depicted using the notation from Figure 8.11. Similarly to the *Experiment Execution* node, this one also requires the specification of two parameters: a batch system manager and an experiment (see Appendix B.5). Using the first one, the instrumentation component can interface with the remote HPC system. The information about the chosen performance analysis tool together with its instrumentation support and the location of the source code of user's application are read from the *Experiment* parameter. Based on these data, the *Instrument Code* node will internally create a shell script on the target HPC system, set up the environment as required by the performance analysis tool and run the instrumentor on the source code of the user. At the end, a log about the re-instrumentation process will be returned. In case an error occurred during the process, this custom node will automatically abort and generate a BPMN exception that can then be manually handled within the workflow.

### 8.4.8. Store Additional Information to an Experiment

Often, there are times when additional, user-specified information must be stored within an experiment in order to coordinate other workflow activities or improve the traceability of the overall analysis and tuning process. For example, this may include baseline characteristics of an application, target optimization criteria, values of different tuning parameters and others. Characteristic for this information is that it is normally not known in advance but is rather generated at runtime. As such, it is important that there is a easy way of storing these user-specified runtime data whenever they get created in a workflow.

Add Exp Extra Info

Figure 8.12.: Custom BPMN node for storing additional information to an experiment

PAThWay implements this functionality using the domain-specific node shown in Figure 8.12. It can be easily plugged in into any performance engineering workflow and takes as input a key-value pair for specifying the data to be stored within the given experiment.

### 8.4.9. Load Custom Performance Results to the Database

Normally, every performance engineering study generates some sort of performance data. Depending on the applied performance tools and the workflow model, this can be high-level profile statistics or detailed results about the behavior of each single function of the analyzed code. Often, this information is left on the target machine and the user has to manually explore it there. However, sometimes it might be more productive to import that performance data to a relational database in order to take advantage of the flexible data querying support of RDBM systems or even share the results with other tools like MATLAB [139] or WEKA [83] for the sake of applying other statistical or data-mining algorithms.

Figure 8.13.: Custom BPMN node for loading custom performance results from an experiment to the internal database

The design of PAThWay foresees the functionality for loading performance results to a database with a specialized domain-specific node (see Figure 8.13). Currently, this BPMN node encapsulates the logic for loading only performance bottlenecks detected by Periscope into the internal database of the framework. This way, users do not have to explore the results manually on the remote machine but can use some of the provided graphical user interfaces to analyze them together with the rest of the collected experiment data. In the future, this custom node can be extended to include support for importing trace data to other tools for storage of raw performance data like the PerfDMF framework discussed in Chapter 5.3.4.

### 8.4.10. Execute a Remote Process

PAThWay defines a generic performance engineering experiment as a single unit of execution of a parallel application on an HPC system with or without any supporting performance analysis tool. Using the previously described BPMN nodes and the other software components of the framework, each such experiment can be autonomously executed and its output, results and runtime environments will be automatically captured.

However, there might be times that an activity in a workflow needs to execute a remote program with a particular set of arguments and simply collect its output for local processing. For instance, many tools that support event tracing performance analysis provide also utilities for extracting summary statistics out of the commonly huge result files. Running such programs on the data generated from an experiment directly on the remote machine and only processing their output locally can be exactly what a workflow needs to do. Encapsulating this activity as a separate PAThWay experiment will be, in this particular case, an overkill and will also fill the internal database with useless, temporary data.

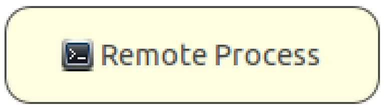


Figure 8.14.: Custom BPMN node for executing an arbitrary remote process

Therefore, a separate domain-specific node is explicitly provided with PAThWay in order to enable the seamless integration of remote processes in the designed workflows. Just like all other custom BPMN nodes, this one is also represented in the palettes of jBPM's workflow editors using the rounded box (see Figure 8.14). As its XML description shown in Appendix B.12 suggests, this nodes defines many input parameters. However, the majority of them are interchangeable. For example, the user has the choice to directly specify the remote machine and its accompanying login details or can simply provide a tag of an HPC system configuration that exists in PAThWay's database. Either way, this node will create a new network connection to the target machine, execute the requested command and finally return its exit status and standard output and error streams. This information can then be mapped to internal workflow variables and processed further by consecutive BPMN nodes.

### 8.4.11. Custom BPMN Node for Interactive Questions

Last but not least, being able to interactively collect information from the user is a fundamental requirement of any tuning workflow. Even though such functionality can also be implemented using the *Human Task* node that is defined by the BPMN standard, it requires the installation and maintenance of additional software components. Moreover, the amount of required configuration steps make it an overkill for the current use-case.

That is why, PAThWay provides a custom node for creating of dialog boxes with generic questions that can be specified at design time, the graphical notation of which is shown in Figure 8.15. As the node's definition from Appendix B.11 shows, it accepts three input parameters: the ques-

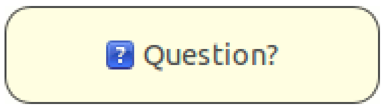


Figure 8.15.: Custom BPMN node for getting interactive response from the user

tion that has to be asked, any default value which should appear as an answer, and the title to be used for the newly generated dialog box. Upon completion of the node, the answer given by the user will be returned which can then be either stored in an internal workflow variable or directly processed.

Chapter 9

# Internal Data Storage Repository

**Contents**

**9.1. Motivation and Design Goals** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **113**
**9.2. Database Entities and Their Application** . . . . . . . . . . . . . . . . . . . . . . . . . **114**
9.2.1. Performance Engineering Experiments . . . . . . . . . . . . . . . . . . . . . 115
9.2.2. Applications . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116
9.2.3. HPC Systems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117
9.2.4. Performance Tools . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118
9.2.5. Historical Notes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 119
9.2.6. Performance Results from Experiments with Periscope . . . . . . . . . . . . 119
**9.3. Object-Relational Mapping using Hibernate** . . . . . . . . . . . . . . . . . . . . . **120**

## 9.1. Motivation and Design Goals

One of the key features of a tool, aiming to support developers during the performance engineering cycle, is its data management component. Each step of a typical workflow can depend on different input data and can generate diverse results ranging from performance measurements, through tuning parameters, to even project documentation.

The internal data storage repository of PAThWay is designed to provide a flexible representation of the data it stores so that it can be easily extended at any time in the future without having to invest many resources in refactoring other internal components. After a detailed analysis of many different alternatives for implementing such a solution, e.g. flat files, custom data format and RDBM systems, it was decided that the framework is going to use an embedded database. This approach has all the advantages and flexibility of relational databases but does not require the installation of any external components. Moreover, this data organization approach enables an easy integration of the framework with other tools such as the Business Intelligence and Reporting Tools (BIRT) plug-in from Eclipse or MATLAB in order to plot and automatically generate custom project reports or to interactively analyze collected data.

## 9.2. Database Entities and Their Application

After deciding on the generic approach for storing PAThWay's data, the second step was to find a specific software product that could be easily integrated in the rest of the framework. Initially, since PAThWay is written in Java and it is distributed as an open-source project, a big subset of the currently available databases were easily filtered out as being incompatible. After this preselection, the list of possible candidates was restricted down to five Java-based databases: HSQLDB (HyperSQL DataBase) [196], Apache Derby [11], Oracle Berkeley DB Java Edition [158] and H2 [153]. Then, in order to find the most suitable one, the functionality of all five of them was carefully analyzed and tested using small prototypes.

As a result of these tests, the open-source H2 database engine was selected for integration due to its very small footprint of about 1 MByte, fast querying of data and support for in-memory databases using either an embedded or a stand-alone execution mode. Furthermore, a convenient web-based management console is also provided out-of-the-box which eases the overall development and testing process. Last but not least, in addition to the standard way of accessing data using Structured Query Language (SQL), the H2 database engine is also natively supported by a persistence framework called Hibernate [43] which is internally used by PAThWay to realize all data manipulation operations and the persistence of objects (see Section 9.3).

One of the last architectural phases after the storage infrastructure was finally chosen was to design PAThWay's underlying data model. This process included analyzing the typical scenarios of using performance engineering workflows and the data needed throughout them in order to properly capture the requirements and create the separate entities for which data has to be collected. The result of this design phase can be seen in the full entity-relationship diagram (ERD) of PAThWay shown in Figure C.7 of Appendix C.1.1.

In the following sub-sections, all the entities of the ERD will be introduced and their typical usage will be discussed. Additionally, references to the full specification of the separate data schemes will be given in each section in order to better illustrate their design.

### 9.2.1. Performance Engineering Experiments

One of the main design goals of PAThWay is to provide support of implicit storage of meta-information about all completed performance engineering experiments. As it can be seen in Figure 9.1, these data is internally maintained using three interlinked SQL tables. They enable an effective organization of the information in order to allow its easy querying and processing. The fundamental data about an experiment is stored in the *Experiment* table. It includes information about experiment's date and time, the chosen application, the applied performance engineering tool and the target HPC system. Additionally, the path to the location where the generated study results are stored is included. The exact description of the columns together with their usage can be found in Appendix C.1.1.

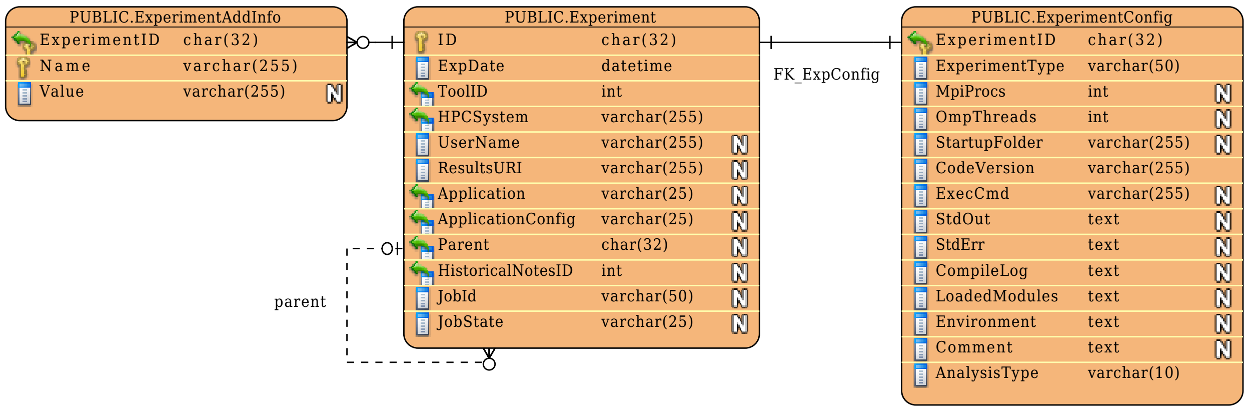


Figure 9.1.: Database scheme for storing experiments

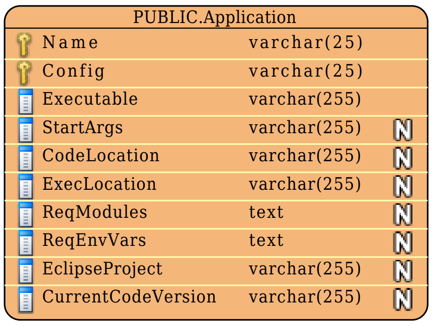
However, storing this information is not enough to provide full end-to-end traceability of a performance engineering process. Among the other very important details about the runtime configuration of an experiment, stored in the *ExperimentConfig* table, are the number of MPI processes and OpenMP threads, the start-up command and the corresponding state of the environment, information about the loaded modules and many others (see Appendix C.1.2). Moreover, as a matter of good practice, the framework will also ask the user to enter a short description/comment before executing any new experiment and store that together with the rest of the configuration. In case the application has to be (re-)instrumented as part of an experiment, the exact log of this process will be additionally saved in the internal database. Last but not least, PAThWay will also capture the standard output and error streams of the underlying parallel job and store those with the runtime settings of the experiment.

As it was discussed in previous chapters, there might be times when a workflow needs to store additional information about an experiment. This can be either dynamically generated by a process activity or manually entered by the user. Due to its generic requirements, it can also be of any data type that is not known in advance. In order to provide flexible support for storing such information, PAThWay also includes an SQL table called *ExperimentAddInfo*. As shown in the full specification of the table in Appendix C.1.4, there can be multiple parameters stored for a single experiment which is also the most common use-case scenario.

### 9.2.2. Applications

An integral part of every performance engineering experiment is the target application. Its runtime behavior has to be carefully analyzed in order to detect potential bottlenecks and decide on a suitable tuning strategy. This process normally starts by annotating the source code of the application with measurement probes provided by the chosen performance analysis tool. Then, the investigated program is started on the target HPC system and varying input and runtime configurations are applied.

In order to enable the automatic execution of the above steps, PAThWay stores additional meta-information about applications in an separate SQL table (see Figure 9.2). Among others, every row includes the start-up arguments of the given application together with a list of all required environment variables and external modules. Additionally, the location of the source code and the final built folder are also specified for the sake of enabling automatic code instrumentation.



PUBLIC.Application

| | | | |
|---|---|---|---|
| 🔑 | Name | varchar(25) | |
| 🔑 | Config | varchar(25) | |
| ▤ | Executable | varchar(255) | |
| ▤ | StartArgs | varchar(255) | N |
| ▤ | CodeLocation | varchar(255) | N |
| ▤ | ExecLocation | varchar(255) | N |
| ▤ | ReqModules | text | N |
| ▤ | ReqEnvVars | text | N |
| ▤ | EclipseProject | varchar(255) | N |
| ▤ | CurrentCodeVersion | varchar(255) | N |

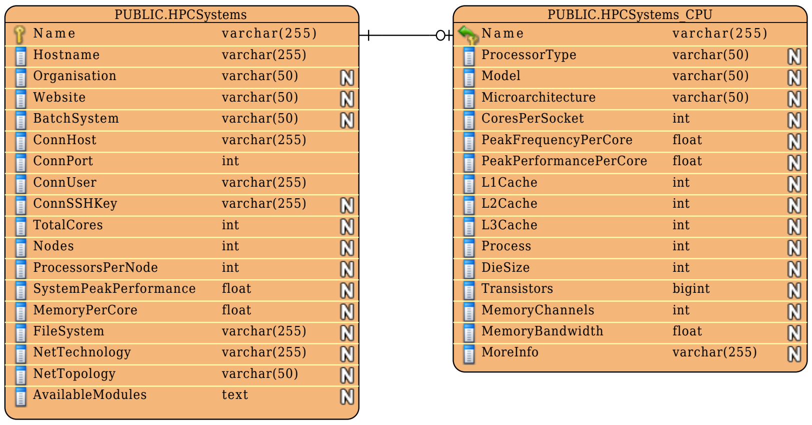
Figure 9.2.: Database scheme for storing the configuration of applications

Moreover, the name of the executable of an application, its command-line arguments and the specified environment variable can be additionally parametrized using a set of predefined placeholders (see Appendix C.1.4). This way, dynamic runtime configurations using, for example, different versions of an executable based on the chosen number of MPI processes and OpenMP threads can be created.

### 9.2.3. HPC Systems

Another key component of PAThWay's data model deals with the storage of configuration data for different HPC systems. This information is internally stored in the *HPCSystems* table and contains, foremost, all the necessary details for accessing a particular machine such as a host name, a user name and an authentication key. Quite often, however, these systems are reachable only through a small set of preconfigured log-in hosts and as such require that users always use specialized connection forwarding techniques like SSH tunneling. Consequently, when configuring a new HPC system, the user can also include an additional address of a machine, i.e. host name and port number, where such a tunnel to the given system exists.

Furthermore, there are other optional fields which can be stored for each HPC system (see Figure 9.3). They include the managing organization of the machine, its overall system configuration and a link to a website describing it in further details. Additionally, due to the fact that most of today's HPC systems operate a dedicated batch system scheduler, its type can also be stored in the *HPCSystems* table in order to allow the framework to automatically configure and submit new performance engineering experiments. In case the given machine has no such scheduling system, the user can choose the default *PAThWay MPICH* entry which will implicitly instruct the framework to use interactive submission and monitoring mode.



| PUBLIC.HPCSystems | | |
|---|---|---|
| Name | varchar(255) | |
| Hostname | varchar(255) | |
| Organisation | varchar(50) | N |
| Website | varchar(50) | N |
| BatchSystem | varchar(50) | N |
| ConnHost | varchar(255) | |
| ConnPort | int | |
| ConnUser | varchar(255) | |
| ConnSSHKey | varchar(255) | N |
| TotalCores | int | N |
| Nodes | int | N |
| ProcessorsPerNode | int | N |
| SystemPeakPerformance | float | N |
| MemoryPerCore | float | N |
| FileSystem | varchar(255) | N |
| NetTechnology | varchar(255) | N |
| NetTopology | varchar(50) | N |
| AvailableModules | text | N |

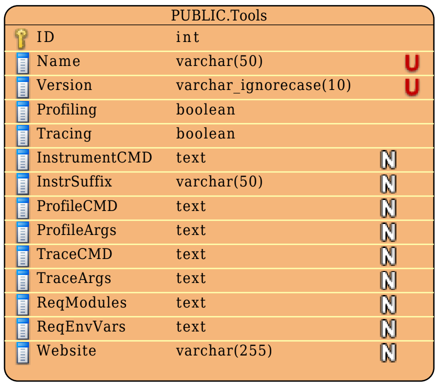
| PUBLIC.HPCSystems_CPU | | |
|---|---|---|
| Name | varchar(255) | |
| ProcessorType | varchar(50) | N |
| Model | varchar(50) | N |
| Microarchitecture | varchar(50) | N |
| CoresPerSocket | int | N |
| PeakFrequencyPerCore | float | N |
| PeakPerformancePerCore | float | N |
| L1Cache | int | N |
| L2Cache | int | N |
| L3Cache | int | N |
| Process | int | N |
| DieSize | int | N |
| Transistors | bigint | N |
| MemoryChannels | int | N |
| MemoryBandwidth | float | N |
| MoreInfo | varchar(255) | N |

Figure 9.3.: Database scheme for storing the configuration of HPC systems

In addition to the host name and log-in details for each system stored in the *HPCSystems* table, PAThWay also provides the support for entering extra information such as the amount of available memory, number and model of CPUs, node-interconnection, peak performance and many others (see Appendices C.1.5 and C.1.6). Having all these key characteristics in one place not only saves the time of getting to know a specific system but can also be used within complex workflows to automate the decision-making process.

### 9.2.4. Performance Tools

One of the last storage entities accessed by almost every performance engineering workflow deals with the configuration of external tools such as runtime monitoring libraries, performance analysis toolkits and other related frameworks. Similarly to the other data units, this information is organized in its own SQL table called *Tools*. When the user wants to add a new tool, PAThWay asks about different facts related to it such as whether it supports only profile-based performance analysis or it is also capable of gathering tracing data, which commands have to be executed for instrumenting the application and how can a new analysis be started. Additionally, tool's runtime requirements and a reference to a website containing further information about it can also be stored in the table (see Appendix C.1.7).



| PUBLIC.Tools | | |
|---|---|---|
| ID | int | |
| Name | varchar(50) | U |
| Version | varchar_ignorecase(10) | U |
| Profiling | boolean | |
| Tracing | boolean | |
| InstrumentCMD | text | N |
| InstrSuffix | varchar(50) | N |
| ProfileCMD | text | N |
| ProfileArgs | text | N |
| TraceCMD | text | N |
| TraceArgs | text | N |
| ReqModules | text | N |
| ReqEnvVars | text | N |
| Website | varchar(255) | N |

Figure 9.4.: Database scheme for storing the configuration of performance tools

What is more, the instrumentation and analysis commands of any performance tool can be specified in a parametrized way in order to enable the integration of runtime settings in them. For example, if a performance analysis tool requires the specification of the number of MPI processes in addition to the application it should analyze, this can be done by using the `{#MPI_PROCS#}` placeholder. Every time when PAThWay prepares to start a new experiment and sees such a string in a start-up command, it will automatically replace that with the correct runtime value. Other placeholders supported by the framework are:

{#TOOL_CMD#} – name of the executable of the given performance tool
{#MPI_PROCS#} – number of MPI processes
{#OMP_THREADS#} – number of OpenMP threads
{#MPI_EXEC#} – MPI execution command (e.g. mpirun, mpiexec, poe)
{#APP_NAME#} – path to the executable for the application
{#APP_ARGS#} – command–line arguments for the application
{#OUTPUT_LOC#} – location of the results from an experiment

### 9.2.5. Historical Notes

Even though the user is always asked to enter a short comment about every performance engineering experiment, sometimes he might want to create a more detailed description about it and store that together with the rest of the experiment's data. For example, this extra information can include a discussion of the collected results or the undertaken tuning steps. It can even contain additional images and links to other resources in order to better illustrate the performed tasks. That is why, PAThWay also provides a Wiki-based module for writing documentation about projects and experiments, the features and design goals of which will be discussed in detail in Chapter 10.

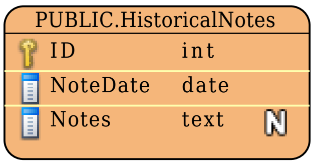


Figure 9.5.: Database scheme for storing user-specified historical notes

Internally, this documentation module also interfaces with PAThWay's database and stores its data in a simple SQL table called HistoricalNotes (see Figure 9.5). Each row represents a single, user-specified note, stored in a textual form using Wiki-based formatting directives and tagged by the date when it was first created. Additionally, every note is implicitly linked to all experiments completed on that day with the aim of providing a more flexible and integrated user interface.

### 9.2.6. Performance Results from Experiments with Periscope

The last set of tables of the internal database of PAThWay are not directly related to the execution of performance engineering workflows. Instead, they are designed to ease the organization and analysis of performance data generated specifically by the Periscope performance analysis toolkit. Due to the high-level, profile-based format of the data, it fits very well with the fundamental idea of a relational database and allows users to take advantage of a structured storage support for performance results and experiments with many querying and processing features. Moreover, once the Periscope results get stored in the internal database, they can also be easily accessed from other data processing tools for further analysis.

As discussed previously in Chapter 5.3.1, Periscope generates two types of files that provide experiments related data: a *SIR* file which contains an outline of the instrumented code regions of a given application and a *PSC* file which includes the collected performance results about its behavior. Internally, the first type of data is stored in a table called *PscRegion* (see Figure 9.6). Each one of its rows specifies uniquely a single code region found in experiment's SIR file and

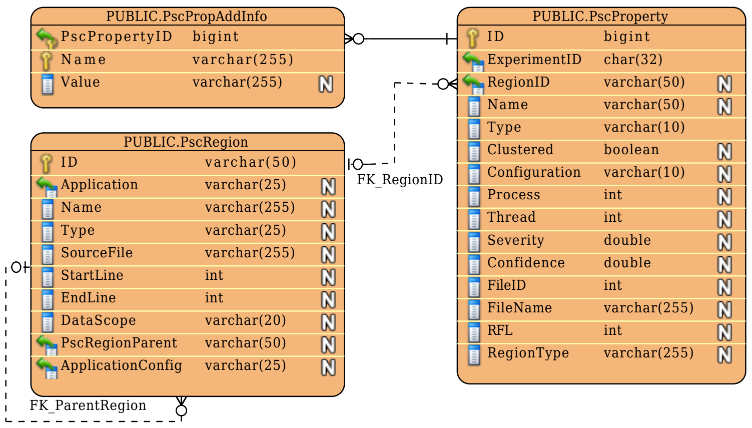


Figure 9.6.: Database scheme for storing performance results generated by Periscope

additionally contains a link to the configuration of the application for which the file was generated. The full definition of all columns of the *PscRegion* table can be found in Appendix C.1.10.

Once the results file from an analysis run with Periscope gets parsed by the framework (e.g. by using the provided domain-specific BPMN node from Chapter 8.4.9), all properties it contains are automatically stored in the *PscProperty* table. This table not only includes all the provided background information about the detected performance bottlenecks, but also links each property to its corresponding code region from the *PscRegions* table (see Appendix C.1.9).

Last but not least, the internal database includes one more SQL table for storing the full range of performance data generated by Periscope. As it can be seen from Figure 9.6, the structure of *PscPropAddInfo* is very generic as it is designed for holding any additional information that a Periscope performance property can provide. For example, this can include the precise number of cache misses related to that bottleneck, a reference to textual tuning parameter or even time series describing the dynamic behavior of the application which led to the generation of the given performance property. As a result, an easy to explore and manipulate structure for storing Periscope's performance results has been successfully created within the framework of PAThWay.

## 9.3. Object-Relational Mapping using Hibernate

Having a database as an internal storage gives a lot of flexibility in the design of the data model of an application. What is more, the existence of established APIs for generic data processing and querying such as Open Database Connectivity (ODBC) [70] and its Java alternative JDBC (Java Database Connectivity) [173] create a powerful environment which provides many different means for accessing the data from both developers and end-users. For example, the very same way a C-based application can use the ODBC API together with a hand-crafted SQL expression in order to fetch the information stored in a database, a Java program can use the JDBC specification without having to worry whether the data is actually stored in a dedicated database cluster or simply in an embedded one.

However, there exists a small drawback of using this approach - it does not fit very well with today's object oriented programming model. These APIs know nothing about the relationship of the data stored in a database and the real objects an application uses. That is why, they also return the information as simple data arrays which have to be manually processed and converted in the final objects before usage. Moreover, the SQL queries have to be created in advance and thus they often end up being linked to a particular database scheme. These issues effectively increase the complexity of using databases as data storage and can have a negative long-term impact on the maintenance of applications.

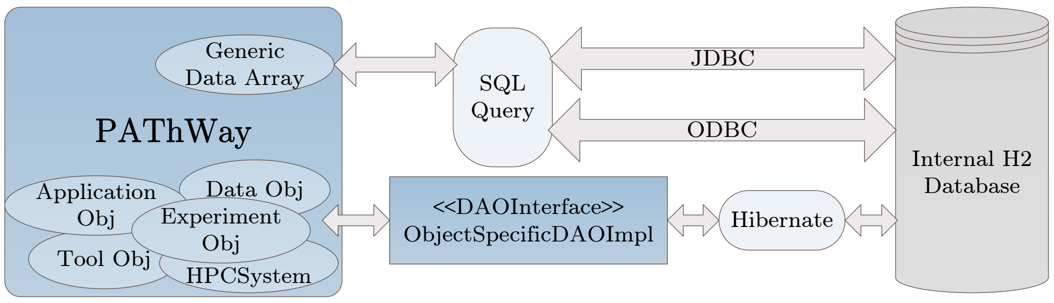


Figure 9.7.: Object-relational mapping of PAThWay's data classes using Hibernate

Fortunately, there are specialized software tools that tackle exactly this inefficiency - the so called Object-relational mapping (ORM) [17] frameworks. In the Java world, Hibernate [43] is the most popular and well-established ORM framework. It not only provides transparent conversion between the data stored in a database and the real application's object, but also includes an object oriented query language, implicit checking for data correctness and automatic database connection management. What is more, the complete configuration of the Hibernate framework together with the mapping between all classes and SQL tables is controlled by a set of XML-based files which are dynamically parsed at runtime. This allows the database scheme to be easily extended without requiring any code modification on the application's side.

The design of PAThWay also adopts Hibernate as an abstraction layer between its database objects and the H2-based database (see Figure 9.7). Internally, this is done using a vertical mapping approach where each database table is represented in the code of the framework as a single Java class (see Figure C.8 in Appendix C.2). Additionally, a generic interface for working with data access objects (DAO) and a factory design pattern are used to further encapsulate all database accesses and create a clean separation between the persistence layer and the actual code of PAThWay. Last but not least, the actual mappings between the plain Java objects representing the internal data and the actual SQL tables are configured using Hibernate's native XML-based configuration files, an example of which can be found in Listing C.3 of Appendix C.2. As a result of these architectural decisions, a very flexible, dynamic and easy to maintain data storage component was created and successfully integrated in PAThWay.

Chapter 10

# Project Documentation Module

**Contents**

**10.1. Motivation and Design Goals** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **123**
**10.2. EclipseWiki and Its Features** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **124**
**10.3. Integration of EclipseWiki with PAThWay** . . . . . . . . . . . . . . . . . . . . . . . . **125**
10.3.1. Modification to EclipseWiki . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 125
10.3.2. Referencing PAThWay's Resources from the Documentation . . . . . . . . . 126
10.3.3. View for Showing Available User's Notes . . . . . . . . . . . . . . . . . . . . . 127
10.3.4. Putting The Pieces Together . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127

## 10.1. Motivation and Design Goals

Having an accessible documentation about requirements, work progress and ideas is very important for the success of every project, not only for those in the area of high-performance computing. There are many tools that specialize in the organization of such data and provide extensive support for maintaining software requirements such as Visual Paradigm [214], Enterprise Architect [191], BOUML [161] and others. However, due to the wide-range of functionality these tools provide, they have a rather steep learning curve. Thus, they are mainly suitable for large, distributed projects with many partners and lots of prerequisites. That is why, they are also not very commonly used in the HPC community. In this area, instead, such information is often highly disorganized and spread across multiple documents on different systems.

Therefore, in order to fill this gap, the design of PAThWay also includes a simple documentation module that can be used together with the workflow environment. It is based on an customized version of the EclipseWiki editor for Eclipse [219] and uses both text files and the internal database as back-end storage. This approach enables the creation of a powerful but yet easy to use project documentation component that fits better to the needs of the HPC community. Its features and implementation details are the subject of discussion of the following sections. First a short overview of the EclipseWiki editor will be given and then the steps involved in its customization for the needs of PAThWay will be discussed.

## 10.2. EclipseWiki and Its Features

EclipseWiki is an open-source plugin for the Eclipse IDE that was specifically designed with the aim to provide a simple documentation environment for both personal and project-based usage. Its development was started in the year 2006 by Channing Walton but many other open-source developers contributed to it in the course of time. Unfortunately, however, the project is not very active anymore and officially supports only an older version of the Eclipse IDE.

Nevertheless, EclipseWiki still provides a flexible and easy to use documentation environment. It builds on top of locally hosted wiki engine and supports three different markup specifications for text formatting: the versatile TWiki [172], the simple SnipSnap [111] and the very first public wiki syntax called Wiki-Wiki-Web (Ward's Wiki) [129]. Using these markup languages, users can apply their own personal style of writing and organizing the documentation without having to adopt a new one. As it will be shortly presented in the next section, different types of enumerations and tables with, for example, performance data can be easily included in the documentation by adding just a couple of extra formatting characters to the text.

Moreover, due to the integration of EclipseWiki with the open-source IDE, users can also insert links to many external resources like files with source code, software projects, web sites, and others. Images can also be used by either directly embedding them in the documentation or linking to their original files. This way, for example, scalability charts and other diagrams showing a specific performance behavior can be easily included to further illustrate the described topics.

Last but not least, a convenient multipage editor is provided with the plug-in which combines a text editor, an interactive viewer of the resulting formatted documentation and a help page with guidelines how to use the chosen wiki markup specification. Additionally, the documentation created with EclipseWiki together with all embedded source files and images can also be exported in HTML format at any time in order to be exchanged with other people or published on a website.

## 10.3. Integration of EclipseWiki with PAThWay

As it was already discussed, an easy to use documentation functionality should definitely be part of every process automation system not only in the area of performance engineering. That is why, during the architectural design of PAThWay different options for the creation of such a documentational module were analyzed and thoroughly tested. Finally, it was decided to integrate the previously described EclipseWiki plug-in for Eclipse as another component of the framework instead of implementing a new one from scratch. In the following sub-sections, a short overview of the process of adapting EclipseWiki for the needs of PAThWay will be given. Then, the resulting module and its user interface will be discussed.

### 10.3.1. Modification to EclipseWiki

EclipseWiki is distributed as an open-source project using the Common Public License (CPL) [91] and as such can be freely modified and extended in any suitable way. Moreover, it can be integrated as a component of both commercial or open-source software and it can then be freely redistributed with it. However, as the license requires, all changes to the original source code should be clearly identified and described. That is why, there modifications are shortly summarized here and additional UML diagrams are included in Appendix D as a reference.

The very first step in adapting EclipseWiki to the runtime environment of PAThWay was to analyze any external dependencies and the structure of the code (e.g. function annotations, usage of APIs, etc.). As it was mentioned in Section 10.2, the mainstream development of this wiki editor was discontinued in the past and, as a result, it heavily depends on version 1.5 of the Java language and the Callisto release of the Eclipse IDE. In the course of time, however, both the Java language and the IDE evolved and some of their APIs and coding requirements changed. Consequently, in order to make EclipseWiki compatible with the newest development, its code was partially refactored and some external libraries were replaced with their newer counterparts.

Due to the fact that EclipseWiki was not designed according to Eclipse's recommendations for extensibility, the second step involved the creation of two Java classes according to an internal specification of EclipseWiki for contributing data regions and shortcuts (see Appendix D.1). This modification allows users to use specially formatted strings in their documentation in order to embed and reference other resources from PAThWay. For example, as it will be discussed later in Section 10.3.2, one can create textual links to the definition of other experiments or can attach another documentation file which is stored in PAThWay's internal database instead of the file system.

The last set of modifications to the Wiki editor was again motivated by its limited extensibility.

As it can be seen from the UML diagrams in Appendix D.2, a couple of other classes had to be changed. This was done in order for the newly contributed data regions and textual links to be accepted by the plug-in and correctly interpreted during runtime. Finally, some other changes to the internal configuration of EclipseWiki and its supported linkage formats were also performed to insure smooth integration with the rest of the PAThWay framework.

### 10.3.2. Referencing PAThWay's Resources from the Documentation

EclipseWiki supports linking to many external resources such as other wiki pages, web sites, source files, images and others. However, with the modifications described in the previous section, it can now also include references to some of the PAThWay's internal data entities. The only requirement for such links to work is the usage of a specific, pre-defined format for defining them. Similarly to all other links supported by EclipseWiki, it is specified in the form

$$< Prefix >:< Command >:< Arguments >$$

where the `Prefix` is an identifier for the underlying processing algorithm, `Command` specifies the exact operation that has to be executed by that algorithm and `Arguments` can be used to supply any additional parameters required by the chosen operation. In the case of PAThWay, these fields should have the following values:

- Prefix: should always be set to `PAThWay`;
- Command: can be either `Exp` for referencing to other experiments or `Note` for linking to internally stored documentation notes;
- Arguments: the corresponding ID for either an experiment or a note. For example, an experiment is defined by its universally unique identifier (UUID) [126] tag and, since only a single note is created for each day, it is referenced using its creation date:

```
PAThWay:Exp:ff8081813b675ba8013b675be8a70000
PAThWay:Note:20121231
```

This way, the Wiki editor will be able to automatically create links to the appropriate PAThWay's data entities and display them as part of the documentation. For example, if a link to an experiment is selected, this will trigger the display of a specialized GUI for exploring all available experiments (see Chapter 11.5) and will highlight the given one. In case of a reference to an internally stored note, the underlying processing algorithm will automatically fetch its contents from the database together with all related performance engineering experiments, format the data and display it in a view similar to that of Figure 10.2. It also has to be mentioned that, in addition to these internally stored notes, users have the ability to create other file-based wiki pages with documentation and reference them by inserting their file names in the text.

### 10.3.3. View for Showing Available User's Notes

Using the above referencing notation, any of the user-specified notes that PAThWay stores in its database can be easily accessed. However, often users might not know when a particular note was initially created in order to properly set a link to it. Moreover, for the sake of enhancing the documentation of any performed by the framework experiment, it also automatically adds new entries to the database upon every new execution. Consequently, many additional notes might actually be available in the internal storage without the user being aware of them.

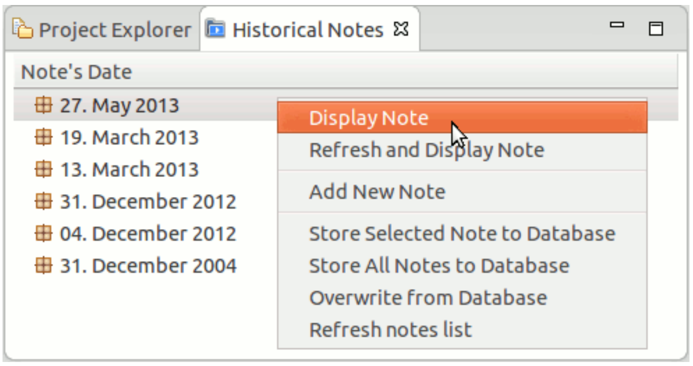
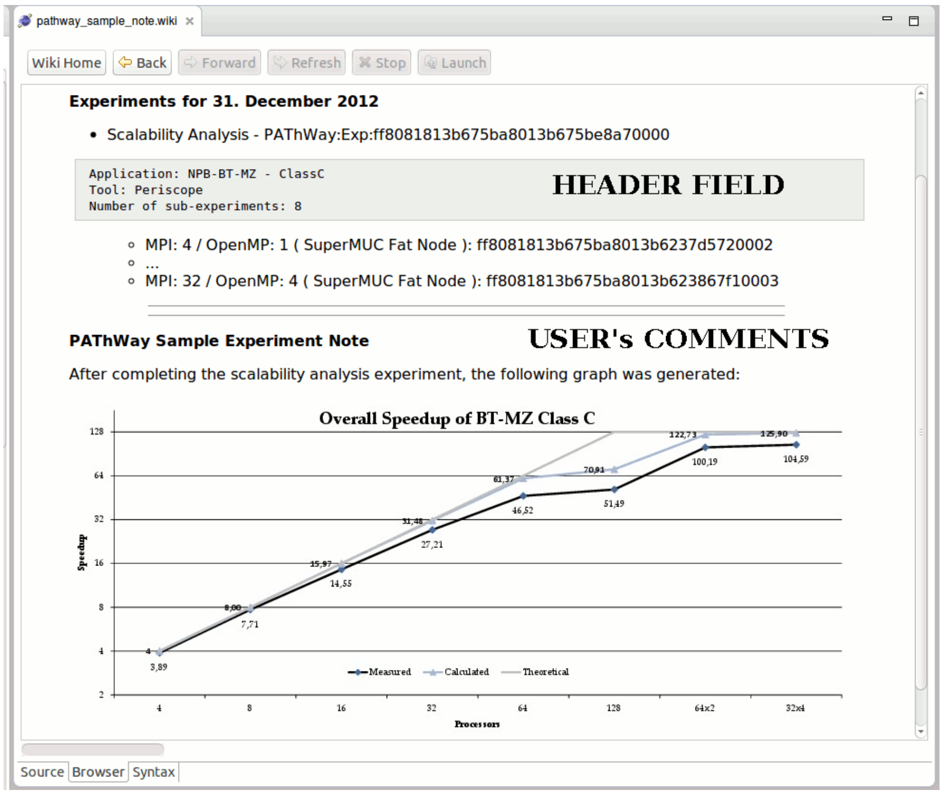


Figure 10.1.: An Eclipse view for exploring and creating user-specified notes.

Therefore, a new view for Eclipse was specifically designed to provide an outline of all available notes in the database. As it can be seen in Figure 10.1, this view shows a sorted representation of the existing notes and, in addition, allows users to add new entries or synchronize the currently displayed ones with the back-end storage. Due to the fact that PAThWay automatically creates a new note for every day when at least one experiment was executed, this view can also serve as a simple overview of the previously completed performance engineering tasks.

### 10.3.4. Putting The Pieces Together

After modifying and extending the EclipseWiki to comply with the newest programming standards and APIs, it was integrated in PAThWay as an additional module for enabling the user to take arbitrary notes while using the framework. An example of how a typical user-specified note looks like when it is visualized in the wiki editor is shown in Figure 10.2. Similarly to all notes that PAThWay generates when running new experiments, this one also consists of two parts: an automatically generated header followed by a field for user's comments and experiment's documentation. The header contains information about all experiments that were executed on day when the note was created. At the same time, the field with user data, which is separated from the header with two horizontal lines, links to an image from another Eclipse project showing a manually generated scalability diagram. All this is represented in plain text form and formatted using only Ward's wiki specification. The exact markup behind the note from the figure can be found in Appendix D.3 as a further reference.



**Experiments for 31. December 2012**

- Scalability Analysis - PAThWay:Exp:ff8081813b675ba8013b675be8a70000

```
Application: NPB-BT-MZ - ClassC
Tool: Periscope
Number of sub-experiments: 8
```

  - MPI: 4 / OpenMP: 1 ( SuperMUC Fat Node ): ff8081813b675ba8013b6237d5720002
  - ...
  - MPI: 32 / OpenMP: 4 ( SuperMUC Fat Node ): ff8081813b675ba8013b623867f10003

**PAThWay Sample Experiment Note**

After completing the scalability analysis experiment, the following graph was generated:

Figure 10.2.: An example of a historical note combining automatically generated information about experiments, user comments and an image with results.

# Chapter 11

# Other Supportive Modules

**Contents**

**11.1. Internal Supportive Modules . . . 129**

**11.2. Revision Control Module . . . 130**

**11.3. Environment Detection Module . . . 131**

**11.4. Runtime Manager and the Parallel Tools Platform Interface . . . 133**

**11.5. Experiments Browser . . . 136**

**11.6. Other Graphical User Interfaces . . . 138**

11.6.1. User interface for Internal Configurations of User's Applications . . . 138

11.6.2. User interface for Internal Configurations of Performance Tools . . . 139

11.6.3. User interface for Internal Configurations of HPC Systems . . . 139

## 11.1. Internal Supportive Modules

The third and bottom layer of the architecture of PAThWay consists of a set of supportive modules. These include a revision control system for tracking the code evolution during the typical performance engineering cycle, an environment detection tool for capturing the exact runtime configuration of experiments and an underlying execution and monitoring system. Common for all of them is that they are purely internal to the framework. As such, they are not directly accessible by the users but only available through the software components of the other layers.

In the following sections, these modules will be described in further details and the motivation behind their design will be shortly discussed. Furthermore, a collection of custom graphical user interfaces for manipulating different parts of the framework will also be presented in order for the reader to get a more complete overview of the PAThWay framework.

## 11.2. Revision Control Module

Because of the high-complexity and repetitive nature of the typical performance engineering cycle presented in Chapter 1.2, having an integrated configuration management system can be of critical importance for the overall success of HPC projects. That is why, a custom revision control component is included as one of the additional supportive modules that form the bottom layer in the architecture of PAThWay (see Chapter 7). Its main function is to track the different source code versions of user's applications which are normally created during the standard iterative tuning process. Storing this information in one place together with all other details about the performed experiments enhances the comparison and the analysis of different application versions and aims to achieve a full end-to-end traceability of the performance engineering process. Furthermore, these data can be analyzed at any later stage to revise the completed process steps in order to plan the next ones and help not to deviate from the predefined project plan.

At the start of every new experiment within the PAThWay framework, the user is always asked to provide a short description/comment about that study. As already discussed in Chapter 9.2.1, this note is then stored together with the configuration of the current experiment into the internal database. In addition to this, however, the provided description is also used by the revision control module in order to tag the current state of the source code. This is done using a software repository which the framework autonomously creates for tracking the code. It has to be mentioned, however, that this internal repository uses a non-default, hidden folder so that it will not interfere with the user's version control system in any way.

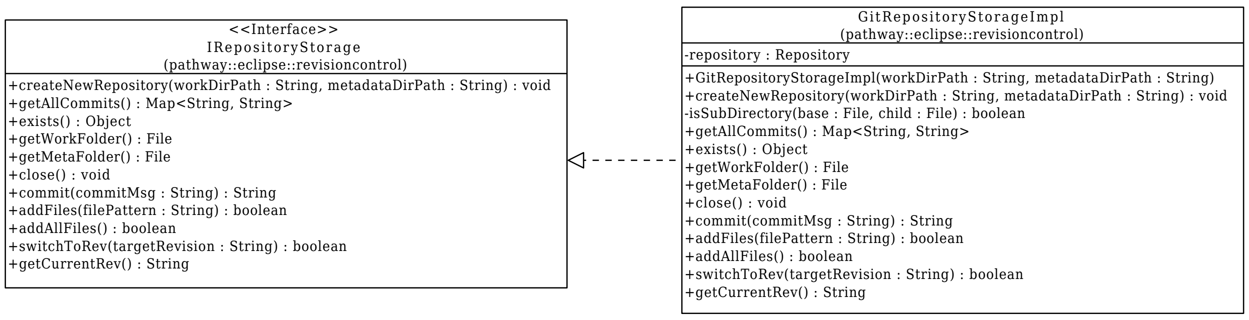

<<Interface>>
IRepositoryStorage
(pathway::eclipse::revisioncontrol)
+createNewRepository(workDirPath : String, metadataDirPath : String) : void
+getAllCommits() : Map<String, String>
+exists() : Object
+getWorkFolder() : File
+getMetaFolder() : File
+close() : void
+commit(commitMsg : String) : String
+addFiles(filePattern : String) : boolean
+addAllFiles() : boolean
+switchToRev(targetRevision : String) : boolean
+getCurrentRev() : String
GitRepositoryStorageImpl
(pathway::eclipse::revisioncontrol)
-repository : Repository
+GitRepositoryStorageImpl(workDirPath : String, metadataDirPath : String)
+createNewRepository(workDirPath : String, metadataDirPath : String) : void
-isSubDirectory(base : File, child : File) : boolean
+getAllCommits() : Map<String, String>
+exists() : Object
+getWorkFolder() : File
+getMetaFolder() : File
+close() : void
+commit(commitMsg : String) : String
+addFiles(filePattern : String) : boolean
+addAllFiles() : boolean
+switchToRev(targetRevision : String) : boolean
+getCurrentRev() : String

Figure 11.1.: Class structure for tracking user's application code

Internally, this component is implemented according to a generic interface which abstracts the underlying revision control system and thus eases future changes to the way PAThWay tracks the source code of user's applications. As it can be seen from Figure 11.1, out of the multiple VCS available nowadays (see Chapter 4), this module currently interfaces with the Git distributed version control system. This functionality is implemented using the pure Java-based implementation of Git called JGit [10]. It provides an extensive support for creating and managing source code repositories and it does not require any external libraries or other software tools. Moreover, since JGit is currently part of the Eclipse IDE, it can be easily interfaced from other plug-ins like PAThWay without contributing any further dependencies.

## 11.3. Environment Detection Module

Implicitly storing the state of the analyzed source code used for each experiment can be a great asset to the developer as it not only increases the transparency and the traceability of the work but also allows one to go back and repeat exactly any past experiment. For example, one might want to test previous code changes on a new compiler or with a new kernel and evaluate if they are still beneficial for the application's runtime or one might need to recreate missing experiment results for an article or a report.

However, often more than just the correct code version is required in order to exactly repeat any past experiments. Other factors such as environment variables, loaded modules, kernel version, etc., can play a major role during execution time. Therefore, PAThWay implements an additional software component that will capture the current runtime environment of every experiment and store it within the experiment's configuration in the internal database (see Chapter 9). This information can then be used by developers to successfully reproduce any old results by using exactly the same configuration as the one from the past.

In order to implement this additional functionality, PAThWay includes a special shell script, the idea of which is to gather diverse information about the runtime environment and store it in a custom format for later processing. Currently, it collects only the exported environmental variables, the loaded modules, the command-line arguments used to start user's application and the precise date and time when it was executed. However, since it is implemented as a bash script [48], it can be easily extended to include other interesting facts about the execution system such as the current version of the kernel, a listing of all running processes, the state of user's resource quota and others.

At the start of every new performance engineering study, PAThWay autonomously uploads this script to the target HPC machine and includes it into the execution sequence of each experiment just before running user's application. In case a supported batch scheduling system is used on the target system, the framework will call the environment detection tool from within the automatically generated batch jobs used for starting the experiments. Otherwise, it will be executed as a separate process immediately before starting user's application. This way, the framework can ensure that the data it collects really represents the execution environment in which the chosen experiment is performed.

Once the environment detection script completes, it stores persistently the collected data in a file, the format of which was carefully chosen in order to be simple to parse and manipulate without requiring extra development resources, be easily portable and, last but not least, be flexible for future extensions. Among the analyzed alternatives, the most suitable data formats were prefiltered down to only four based on their applicability to the given requirements: the plain old INI format which is commonly used on Windows machines, the flexible Protocol Buffers from Google [79], the Extensible Markup Language (XML) [27] and the JavaScript Object Notation (JSON) [49] standard. Using small prototypes in the form of both shell scripts and Java programs, they were all examined and the JSON format was finally chosen. It provides a rather simple, human-readable data specification format as shown in Listing 11.1. At the same time, it has less overhead than XML and it is more commonly used than the Protocol Buffers.

Listing 11.1: An example of the information about the runtime environment of an experiment as generated by the environment detection script of PAThWay

```
{
"dateTime":"01.06.2013 12:53:28",
"executable":"SCOREP_EXPERIMENT_DIRECTORY=Exp.201306011253.32x4 srun_ps",
"arguments":"-x -n 32 /home/user1/bt-mz_C.32.scorep",
"environment":{
        "PAPI_LIB":"-L/opt/papi/lib/ -lpapi -lpfm",
        "OMP_NUM_THREADS":"4",
        "SCOREP_FILTERING_FILE":"scorep.filt",
            ...
        "PERISCOPE_ROOT":"/home/user1/workspace/Periscope",
        "LRZ_SYSTEM":"Cluster",
        "_pathway_env":"end"
    },
"modules":[
        "mkl/11.0",
        "fortran/intel/12.1",
        "ccomp/intel/12.1",
        "mpi.parastation/5.0/intel",
            ...
        "periscope/1.5",
        "papi/4.9",
        "_pathway_modules_end"
        ]
}
```

Finally, once the data format was specified, the last step of the design and the implementation of the environment detection module was to choose a suitable approach of integrating the JSON format into the rest of the framework as it is not natively supported by the Java programming language. However, due to JSON's popularity, there are a couple of different open-source libraries that implement it. Among them, the most flexible and stable one is provided by the Gson [78] project. It allows data specified in this format to be directly converted into Java objects without having to first create specialized parsers. Furthermore, it includes serialization of complete object hierarchies without requiring any additional code notations or export functions. As a result, an efficient and highly extensible data format and a control module for capturing runtime information about the execution environment of performance engineering experiments were implemented and successfully integrated within the PAThWay framework.

## 11.4. Runtime Manager and the Parallel Tools Platform Interface

As part of every performance engineering study, PAThWay uses a specialized runtime manager for controlling both local and remote experiments that are part of it. It serves as a connection point between an underlying execution and monitoring system and the externally visible *Run Experiment* domain-specific BPMN node discussed in Chapter 8.4. This module is responsible for interpreting runtime configurations of experiments which are normally created during the execution of performance engineering workflows and setting up parallel jobs for performing the necessary analysis and tuning steps. Additionally, it is also in charge of maintaining network connections with target HPC machines and, eventually, collecting the results once they become available. Furthermore, it is designed to compound other components of the framework such as the documentation module, the internal database, the environment detection module and others in order to create PAThWay's integrated infrastructure for running HPC workflows.

Internally, the runtime manager of PAThWay interfaces with the Parallel Tools Platform (PTP)

[222] to provide a platform-independent access to both local and remote computational resources and thus allow for greater adaptability of the tool to new exection environments. The biggest advantage of integrating PTP is that it provides a highly configurable resource management infrastructure. This functionality is based on the dynamic parsing of XML configuration files using the Java Architecture for XML Binding (JAXB) [114] and the subsequent creation of new resource managers out of those files. As a result, users can easily create their own custom resource orchestrators without having to write extra code or requiring any additional Eclipse plugins (see Chapter 5.5.2).

Based on these JAXB resource managers, this internal component of PAThWay is designed to provide a generic approach for starting and monitoring interactive and batch jobs for executing performance engineering processes in either parallel or sequential modes. Using customized resource manager definitions, the framework can automatically configure and submit new experiments based on the user-specified requirements and can additionally call its environment detection tool as part of the process in order to capture the exact runtime environment of those experiments.

Currently, three different resource manager types are supported with the framework: two batch system schedulers, i.e. the open-source *SLURM* [227] resource manager and IBM's *LoadLeveler* [112], and an interactive starter based on the *MPICH* [28] parallel runtime environment. However, extending the support for a new system requires just the creation of a simple configuration entry in the HPCSystems table of the internal database. As previously discussed in Chapter 9.2.3, as a bare minimum it has to contain the host name and log-in details for new machine and the type of the underlying scheduling system. However, it can also include additional information about the underlying hardware and software platforms. In case the specified scheduling system is not currently supported by the framework, users have to additionally write a new JAXB configuration file and register it with PAThWay.

Such a resource manager configuration can be created using one of the provided templates or it can be written from scratch. Either way, it is important that a set of internal variables are defined as they are used by the framework to control different aspects of the runtime. Additionally, PAThWay's environment detection script has to be automatically staged-in to the target HPC system on every experiment. This is accomplished with the help of the managed files functionality of PTP's resource managers and it can be configured using the `managed-files` XML node as shown in Listing 11.2. Furthermore, the process start-up techniques, i.e. either the batch script template typically included in such configuration files or the specified commands for interactive application starters, have to be adapted accordingly (see Listing E.1).

Listing 11.2: Extra runtime configuration used by PAThWay for every JAXB-based resource manager that is integrated in the framework

```
<attribute name="pathway.job.name" visible="true"/>
<attribute name="pathway.modules" visible="true"/>
<attribute name="mpiNumberOfProcesses" visible="true"/>
<attribute name="ompNumberOfThreads" visible="true"/>

<attribute name="pathway.get-env.script" visible="true">
        <default>resources/getEnv.sh</default>
</attribute>

<attribute name="pathway.toolrun" visible="true" type="boolean">
        <default>false</default>
</attribute>

<managed-files>
        <file-staging-location>${ptp_rm:directory#value}</file-staging-location>
        <file name="get-env-script">
            <path>${ptp_rm:pathway.get-env.script#value}</path>
        </file>
</managed-files>
```

Furthermore, PAThWay's internal resource manager also abstracts the Remote Tools component of PTP in order to provide semi-transparent access to the file system of any of the configured HPC systems. This way, for example, the framework is able to fetch performance results from remote systems or move any necessary input files to the target machines before each experiment. Additionally, by interfacing with this component of the Parallel Tools Platform, the framework can also execute diverse processes such as code instrumentation and parallel post-processing of data directly on the remote machines. As a result of this approach, a platform-independent access to any computational resource is enabled in a generic and portable way in order to allow PAThWay to adapt easily to many different environments and support the user throughout the overall tuning cycle.

## 11.5. Experiments Browser

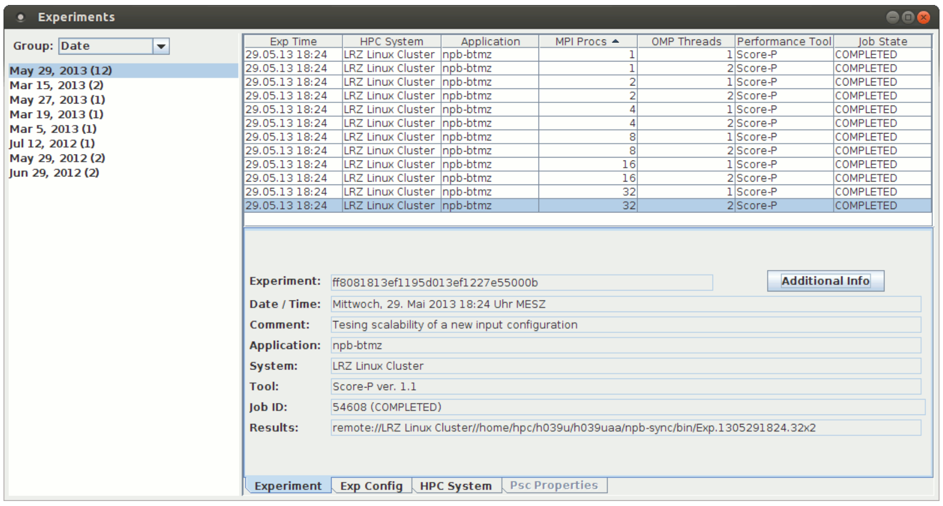
As described in the previous chapters, PAThWay consists of three architectural layers and multiple software components that interoperate in order to create a flexible framework for designing and managing performance engineering workflows. These modules range from simple graphical BPMN nodes through a database to a complex execution and monitoring system. They are designed to abstract different performance analysis and tuning tasks and to automatically create high-level experiments out of them. The later ones can then be autonomously executed by the framework and, as discussed in Chapter 9.2, information about their runtime behavior will be implicitly collected and stored in the internal database.

Once these diverse data about experiments gets stored in the database, it can be access from external applications using any of the public data-access APIs such as ODBC and JDBC. However, there is no predefined graphical user interface for browsing and analyzing the collected data. That is why, an external GUI was designed and implemented as part of PAThWay in order to fill this gap. As it can be seen in Figure 11.2, it consist of three interlinked panels that represent all the internal data about experiments in a integrated and effective way. The left side of the GUI features a listing of all available experiments grouped according to a certain criterion such as execution date, chosen HPC system, applied performance engineering tool, selected user's application and others.

Whenever a group is selected from the left panel, the top right side of the GUI will be automatically updated to include a table with an outline of all experiments in that particular group. Upon selecting a single entry from that table, the tabs at the bottom will consequently display the full information about it stored in the internal database. For example, a summary including experiment's execution date and time, the chosen user's application, the performance engineering tool, the target HPC system and others will be shown in the first tab (see Figure 11.2a).

At the same time, a more detailed information about the runtime environment together with three logs for experiment's standard output, error and source code compilation will be displayed in the second tab as shown in Figure 11.2b. Further examples of the separate GUIs for visualizing these additional data can be found in Appendix E.2.

Moreover, the underlying configuration for the HPC system used within the selected experiment will be presented in the third tab of PAThWay's browser (see Figure 11.2c). This information aims at providing an in-place overview of the hardware and software setup of the target machine in order to help users get further insight into the overall execution of the given performance engineering study.

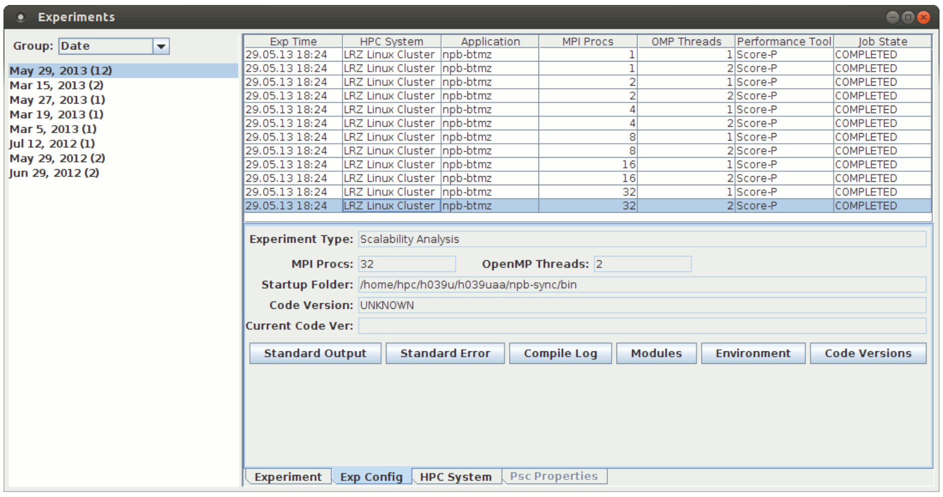


May 29, 2013 (12)
Mar 15, 2013 (2)
May 27, 2013 (1)
Mar 19, 2013 (1)
Mar 5, 2013 (1)
Jul 12, 2012 (1)
May 29, 2012 (2)
Jun 29, 2012 (2)

| Exp Time | HPC System | Application | MPI Procs ▲ | OMP Threads | Performance Tool | Job State |
|---|---|---|---|---|---|---|
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 1 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 1 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 2 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 2 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 4 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 4 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 8 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 8 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 16 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 16 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 32 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 32 | 2 | Score-P | COMPLETED |

Experiment: ff8081813ef1195d013ef1227e55000b
Date / Time: Mittwoch, 29. Mai 2013 18:24 Uhr MESZ
Comment: Tesing scalability of a new input configuration
Application: npb-btmz
System: LRZ Linux Cluster
Tool: Score-P ver. 1.1
Job ID: 54608 (COMPLETED)
Results: remote://LRZ Linux Cluster//home/hpc/h039u/h039uaa/npb-sync/bin/Exp.1305291824.32x2

(a) The first tab of the browser displaying general information about the selected experiment.

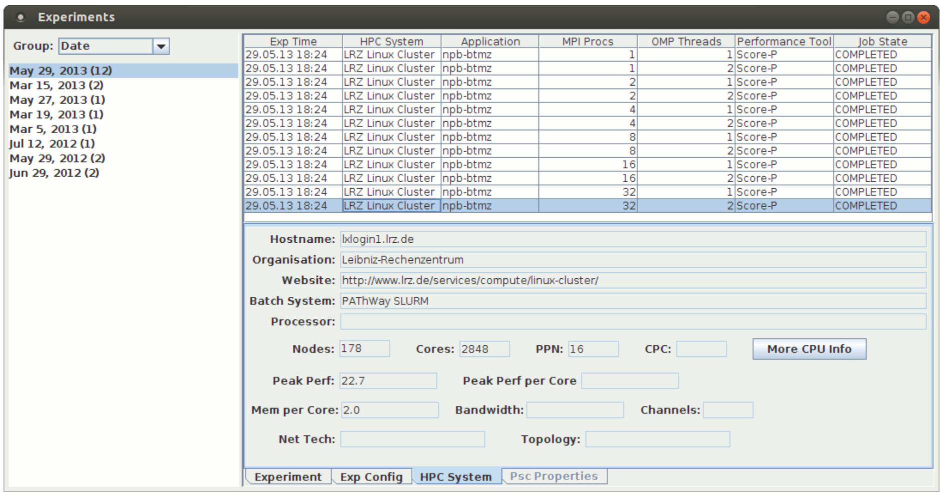


May 29, 2013 (12)
Mar 15, 2013 (2)
May 27, 2013 (1)
Mar 19, 2013 (1)
Mar 5, 2013 (1)
Jul 12, 2012 (1)
May 29, 2012 (2)
Jun 29, 2012 (2)

| Exp Time | HPC System | Application | MPI Procs | OMP Threads | Performance Tool | Job State |
|---|---|---|---|---|---|---|
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 1 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 1 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 2 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 2 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 4 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 4 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 8 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 8 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 16 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 16 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 32 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 32 | 2 | Score-P | COMPLETED |

Experiment Type: Scalability Analysis
MPI Procs: 32    OpenMP Threads: 2
Startup Folder: /home/hpc/h039u/h039uaa/npb-sync/bin
Code Version: UNKNOWN
Current Code Ver:

(b) A tab showing extra data such as standard output and error, runtime environment, and others.



May 29, 2013 (12)
Mar 15, 2013 (2)
May 27, 2013 (1)
Mar 19, 2013 (1)
Mar 5, 2013 (1)
Jul 12, 2012 (1)
May 29, 2012 (2)
Jun 29, 2012 (2)

| Exp Time | HPC System | Application | MPI Procs | OMP Threads | Performance Tool | Job State |
|---|---|---|---|---|---|---|
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 1 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 1 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 2 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 2 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 4 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 4 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 8 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 8 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 16 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 16 | 2 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 32 | 1 | Score-P | COMPLETED |
| 29.05.13 18:24 | LRZ Linux Cluster | npb-btmz | 32 | 2 | Score-P | COMPLETED |

Hostname: lxlogin1.lrz.de
Organisation: Leibniz-Rechenzentrum
Website: http://www.lrz.de/services/compute/linux-cluster/
Batch System: PAThWay SLURM
Processor:
Nodes: 178    Cores: 2848    PPN: 16    CPC:
Peak Perf: 22.7    Peak Perf per Core
Mem per Core: 2.0    Bandwidth:    Channels:
Net Tech:    Topology:

(c) A tab giving information about the HPC system on which the selected experiment was executed.

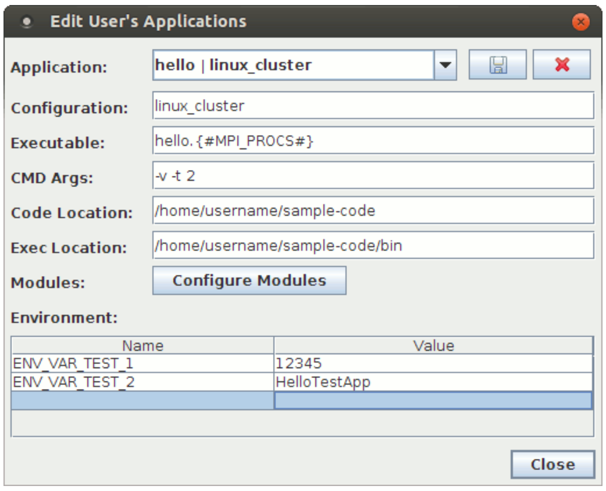
Figure 11.2.: An overview of PAThWay's experiments browser

## 11.6. Other Graphical User Interfaces

The design of PAThWay uses a centralized database system for holding configurations of applications, performance tools and HPC systems and additionally runtime information about experiments. However, similarly to the motivation behind the experiments browser presented in the previous section, the database on its own does not provide any convenient interface for editing the internally stored settings of workflow components or for adding new ones. Therefore, the framework implements three custom graphical user interfaces that are specifically designed to enable these tasks and allow the users to adapt the configuration of PAThWay to their own needs. In order for the reader to get first impressions, these GUIs will be shortly presented in the following sub-sections.

### 11.6.1. User interface for Internal Configurations of User's Applications

The first user interface for manipulating the settings stored in the internal database is the `Edit User's Applications` view. It allows users of the framework to create new configurations for applications or edit existing ones. As it can be seen from Figure 11.3, this GUI enables the specification of all program parameters as defined by the data model from Chapter 9.2.2. These include not only the required command-line arguments, the environmental variables and the external modules, but also the location of application's source code and the start-up folder in order to enable automatic instrumentation and execution. As a result, the users can conveniently edit or insert new configurations of their codes which can then be easily selected from any workflow model using the `Application Selection` domain-specific node presented in Chapter 8.4.1.


Edit User's Applications

| | |
|---|---|
| Application: | hello \| linux_cluster |
| Configuration: | linux_cluster |
| Executable: | hello.{#MPI_PROCS#} |
| CMD Args: | -v -t 2 |
| Code Location: | /home/username/sample-code |
| Exec Location: | /home/username/sample-code/bin |
| Modules: | Configure Modules |

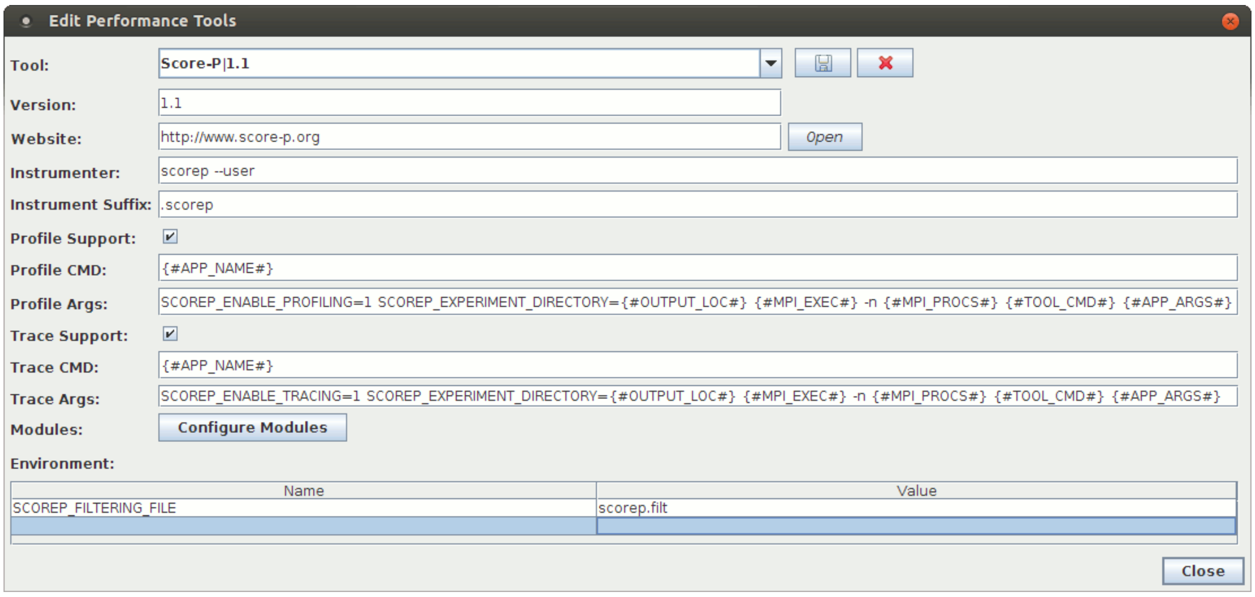
Environment:

| Name | Value |
|---|---|
| ENV_VAR_TEST_1 | 12345 |
| ENV_VAR_TEST_2 | HelloTestApp |

Figure 11.3.: User interface for editing available configurations of user's applications

### 11.6.2. User interface for Internal Configurations of Performance Tools

The second supportive GUI enables the management of configurations of different performance tools. Similarly to the previous one, this user interface allows the integration of new performance engineering tools in the framework by specifying their runtime requirements and features. Following the internal data model for such a performance tool presented in Chapter 9.2.4, this GUI enables the user to enter tool's standard instrumentation command together with its arguments and whether it supports only profile-based analysis or it is able to collect detailed event traces from the applications (see Figure 11.4). In both last cases, the exact commands for starting a new analysis have to be also specified. However, as defined by the data model, these can be additionally parametrized using placeholders for runtime parameters in order to allow their more flexible specification. Similarly to the UI for editing user's applications, the execution environment and the set of external modules required by the given tool can also be set. Finally, with the help of the custom BPMN node described in Chapter 8.4.4, such configurations of tools can be seamlessly integrated in any of PAThWay's workflow models.



| | |
|---|---|
| Tool: | Score-P\|1.1 |
| Version: | 1.1 |
| Website: | http://www.score-p.org |
| Instrumenter: | scorep --user |
| Instrument Suffix: | .scorep |
| Profile Support: | ✓ |
| Profile CMD: | {#APP_NAME#} |
| Profile Args: | SCOREP_ENABLE_PROFILING=1 SCOREP_EXPERIMENT_DIRECTORY={#OUTPUT_LOC#} {#MPI_EXEC#} -n {#MPI_PROCS#} {#TOOL_CMD#} {#APP_ARGS#} |
| Trace Support: | ✓ |
| Trace CMD: | {#APP_NAME#} |
| Trace Args: | SCOREP_ENABLE_TRACING=1 SCOREP_EXPERIMENT_DIRECTORY={#OUTPUT_LOC#} {#MPI_EXEC#} -n {#MPI_PROCS#} {#TOOL_CMD#} {#APP_ARGS#} |
| Modules: | |

Environment:

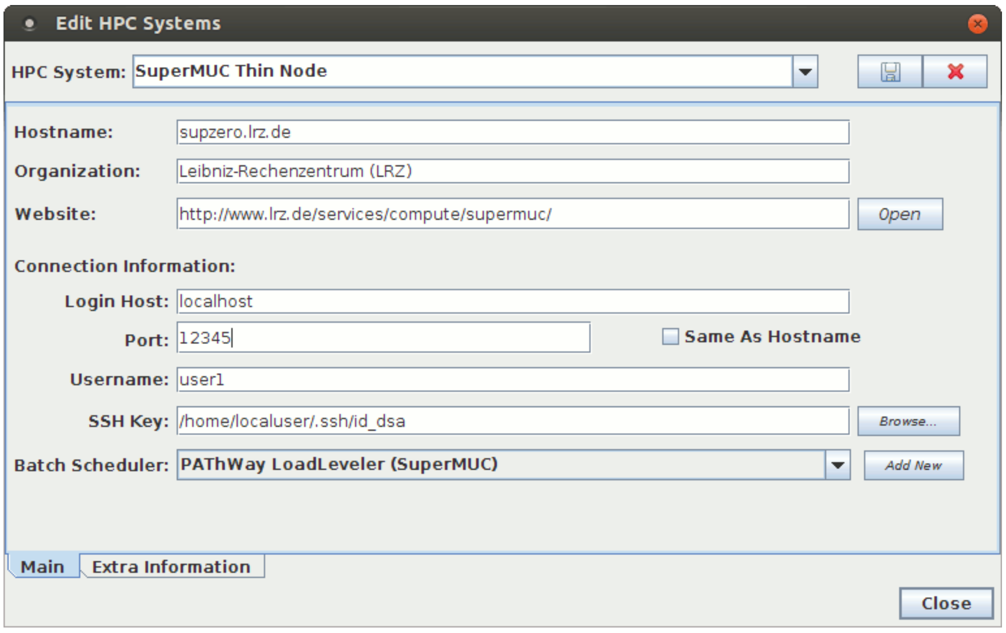
| Name | Value |
|---|---|
| SCOREP_FILTERING_FILE | scorep.filt |

Figure 11.4.: User interface for editing available configurations of performance tools

### 11.6.3. User interface for Internal Configurations of HPC Systems

The last graphical user interface provided by PAThWay for modifying internally stored configuration data is the `Edit HPC Systems` view. As it name suggests, it enables users to integrate new

HPC systems in the framework so that they can be used for execution of performance engineering experiments. This GUI is designed to resemble the look of the previous two in order to create a consistent overall appearance of the user interfaces. However, due to the larger amount of data that can be provided about every HPC system, this one consists of two additional tabs.


Edit HPC Systems

HPC System: SuperMUC Thin Node

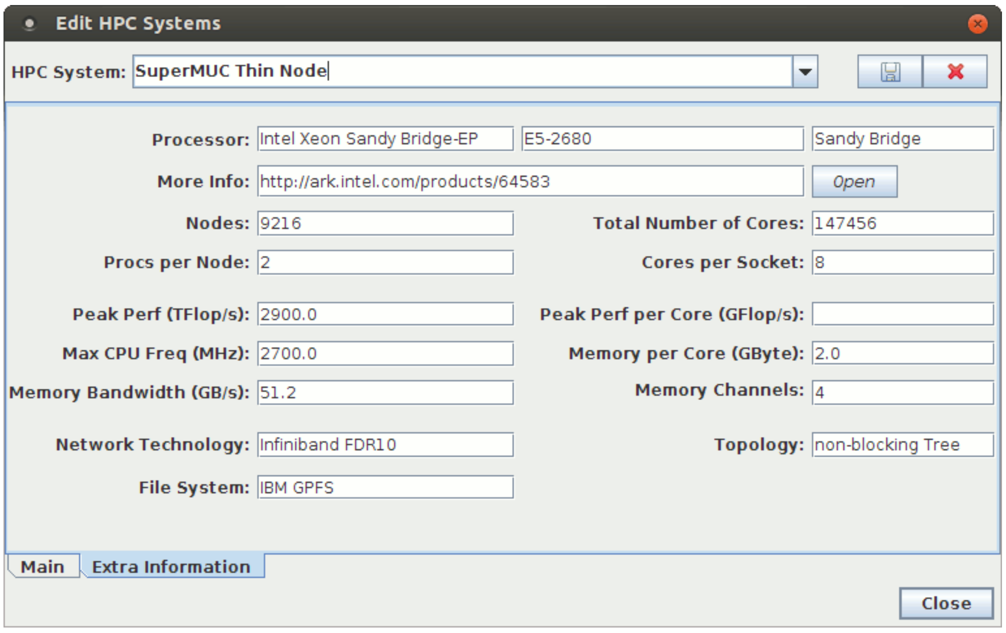
Hostname: supzero.lrz.de
Organization: Leibniz-Rechenzentrum (LRZ)
Website: http://www.lrz.de/services/compute/supermuc/

Connection Information:
Login Host: localhost
Port: 12345
☐ Same As Hostname
Username: user1
SSH Key: /home/localuser/.ssh/id_dsa
Batch Scheduler: PAThWay LoadLeveler (SuperMUC)

Figure 11.5.: The first tab of the GUI showing general information about the given HPC system.

As it can be seen from Figure 11.5, the first one asks the user to enter general information for the given system such as host name, log-in details and information about the underlying scheduling infrastructure that is available on that machine. These settings are mandatory as they are internally required by PAThWay in order for it to be able to create and configure new experiments and automatically execute them on the given machine on behalf of the user.

Figure 11.6 shows the second tab of this GUI which allows users to specify additional information about the underlying hardware and software configuration of the HPC system. These data is purely optional as it is not directly used by the framework. Nevertheless, it can provide valuable insights about machine and the runtime settings of experiments and thus create a more integrated user's experience.



| | | | |
|---|---|---|---|
| HPC System: | SuperMUC Thin Node | | |
| Processor: | Intel Xeon Sandy Bridge-EP | E5-2680 | Sandy Bridge |
| More Info: | http://ark.intel.com/products/64583 | | |
| Nodes: | 9216 | Total Number of Cores: | 147456 |
| Procs per Node: | 2 | Cores per Socket: | 8 |
| Peak Perf (TFlop/s): | 2900.0 | Peak Perf per Core (GFlop/s): | |
| Max CPU Freq (MHz): | 2700.0 | Memory per Core (GByte): | 2.0 |
| Memory Bandwidth (GB/s): | 51.2 | Memory Channels: | 4 |
| Network Technology: | Infiniband FDR10 | Topology: | non-blocking Tree |
| File System: | IBM GPFS | | |

Figure 11.6.: The second tab showing additional information about the system.

# Part III.

# Performance Engineering Workflows in Action

Chapter 12

# Scalability Analysis Workflow

**Contents**

**12.1. Scalability Analysis Process** . . . . . . . . . . . . . . . . . . . . . . . . . . . . 145
**12.2. Workflow Model** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 146
**12.3. LRZ Linux Cluster** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 147
**12.4. NAS Parallel Benchmarks Multi-Zone** . . . . . . . . . . . . . . . . . . . . . . . . 147
**12.5. Workflow Execution** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 148
**12.6. Results Exploration** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 149

## 12.1. Scalability Analysis Process

In this section, the functionality of PAThWay will be used to automate the execution of one of the most popular performance engineering processes among the HPC community - the scalability analysis. As it was already discussed in Chapter 6.2, this workflow deals with the execution of an algorithm (also called computational kernel) or a whole parallel application using an increasing number of processors. This process is not only fundamental for the evaluation of the design of parallel algorithms, but also for the estimation of their suitability to a particular software and hardware combination.

The most common performance metric during such an experiment is execution time. It can be measured for the application as a whole or for its code regions. These metrics are then subsequently used to compute the speedup and the efficiency of the code according to Equations (6.1) and (6.2) and serve as crucial indicators for planning further optimization steps.

## 12.2. Workflow Model

A predefined model for investigating the strong scalability of applications is provided with PAThWay. This workflow interactively guides the performance engineer in creating, configuring and executing such studies. It starts with the selection of an application and the configuration of the number of processors (see Figure 12.1). Then, the framework asks the user to select a given performance engineering tool and a machine which will be used throughout the experiment.

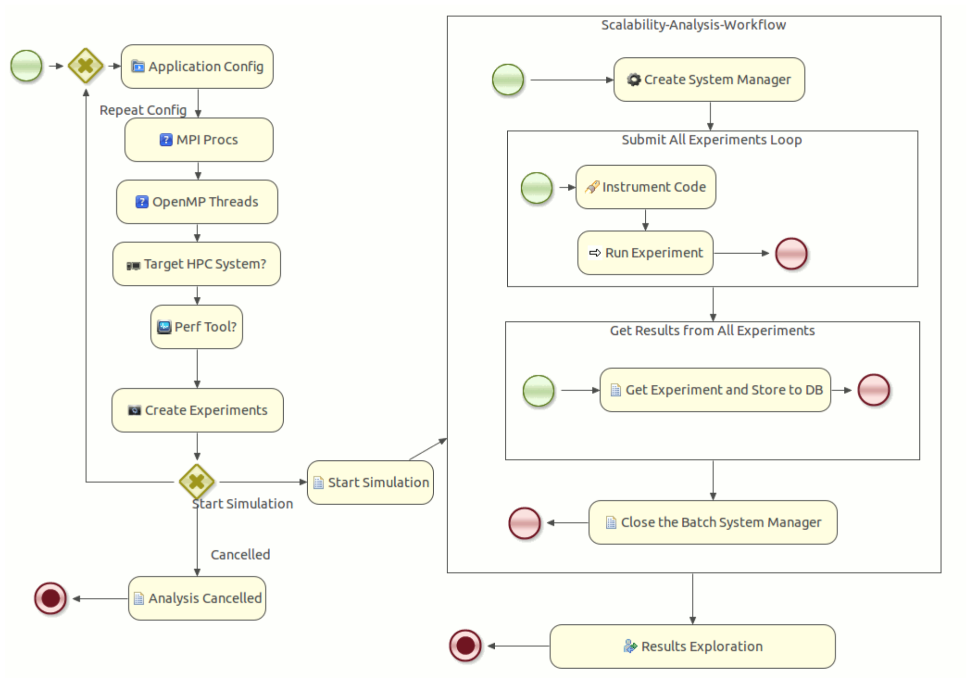


Figure 12.1.: Scalability analysis workflow model

After the interactive setup of the runtime environment, a set of experiments will be created based on the provided settings. Each sub-experiment will represent a single step of the scalability analysis, i.e., a combination of MPI processes, OpenMP threads and environment variables. Next, the tool will create a remote connection to the chosen HPC system and will instrument the source code of the application if required. Additionally, a separate batch script will be created for each step of the study using the information provided by the user. This script will not only start the analyzed code, but it will also trigger the environment detection tool in order to persist all runtime settings. After executing all sub-experiments of the scalability study, PAThWay will create entries for each one of them in the internal database in order to store their runtime information. Finally, it will provide a list of all performed experiments together with their results to the user for further investigation.

## 12.3. LRZ Linux Cluster

The MPP segment of the linux cluster at the Leibniz Supercomputing Centre (LRZ) in Garching, Germany is selected as a target machine for this scalability study. It consists of 178 compute nodes with 2 sockets per node and 8 cores per socket. The nodes have AMD Opteron 6128 HE octo-core processors which have three levels of cache: 128KB L1 (64KB data and 64KB instruction cache) and 512KB L2 cache per core and 2 x 6MB L3 cache shared among all cores. They can process 4 instructions per clock cycle and support 1 hardware thread per core (i.e. no hyperthreading). The total aggregated operating memory of the cluster is 2.8 TBytes with up to 16 GBytes per node. MPP uses a quad data rate Infiniband switched fabric interconnect with an effective transmission rate of 8 Gbit/s in each direction and up to 1.07 microseconds end-to-end MPI latency. The aggregate peak performance of the cluster is 22.7 TFlop/s.

The batch job scheduling on the MPP cluster is controlled from a SLURM [227] scheduler. Interactive parallel jobs are also supported but limited to only 8 simultaneous sessions in total.

## 12.4. NAS Parallel Benchmarks Multi-Zone

The NAS Parallel Benchmarks (NPB) [15] are one of the most common benchmark suites meant for evaluation of software and hardware in the area of high-performance computing. The suite contains multiple highly iterative codes solving a broad spectrum of numerical problems with unstructured adaptive computational grids and parallel I/O. However, the most popular version of NPB is its multi-zone benchmark which targets the solution unsteady, compressible Navier-Stokes equations in three spacial dimensions. Similarly to other scientific computing codes, the discretization domain used within the benchmarks is split on among the processors to achieve a coarse-grain parallelization. This leads to a lot of communication between participating processors in order to exchange locally computed boundary values after every iteration.

The numerical problem which these benchmarks solve exposes different levels of parallelism. This is, however, not exploited by the pure MPI and OpenMP versions of NPB and can easily lead to load imbalance situations. Consequently, a new hybrid version called NPB Multi-Zone [108] was developed by the NASA Advanced Supercomputing Devision. It exploits both programming models to create a multi-level parallelization of the code. It separates the discretization domain in multiple fine-grain zones and then uses load balancing techniques to distribute these zones for the sake of improving the utilization of the computational resources. Specifically, OpenMP is used within a zone to parallelize the computation of a local solution and MPI is applied on a coarser level to exchange the values of overlapping areas between processors. This way, a global solution

Table 12.1.: NAS Parallel Benchmark Multi-Zone Problem Classes [108]

| Problem Class | Domain Dimensions | Number of Zones BT-MZ | SP-MZ | LU-MZ | Total Memory |
|---|---|---|---|---|---|
| S | 24 x 24 x 6 | 2 x 2 | 2 x 2 | 4 x 4 | 1 MB |
| W | 64 x 64 x 8 | 4 x 4 | 4 x 4 | 4 x 4 | 6 MB |
| A | 128 x 128 x 16 | 4 x 4 | 4 x 4 | 4 x 4 | 50 MB |
| B | 304 x 208 x 17 | 8 x 8 | 8 x 8 | 4 x 4 | 200 MB |
| C | 480 x 320 x 28 | 16 x 16 | 16 x 16 | 4 x 4 | 800 MB |
| D | 1632 x 1216 x 34 | 32 x 32 | 32 x 32 | 4 x 4 | 12800 MB |

for the whole domain can be efficiently computed.

The NPB Multi-Zone benchmarks provide three different algorithms for the solution of the Navier-Stokes equations: Lower-Upper symmetric Gauss-Seidel (LU-MZ), Scalar Penta-diagonal (SP-MZ) and Block Tri-diagonal (BT-MZ). As shown in Table 12.1, six problem sizes are provided with the benchmarks. In the current performance engineering study, the second biggest *Class C* will be used to evaluate the scalability of the BT-MZ algorithm.

## 12.5. Workflow Execution

The very first step before starting this study is the creation of an Eclipse project to store the source code of NPB-MZ. Even though this step is not mandatory as PAThWay can also work independently of projects, it allows to better utilize all the powerful code development and organization features of Eclipse. As a project type, a synchronized project is chosen due to its suitability for cross-platform development. The advantage of such a project is that all local files are mirrored on the remote systems and are internally kept synchronized. This is internally achieved using a hidden, automatically maintained Git-based repository which functions in parallel to user's own source code repository. Moreover, building of the project will be done automatically on the remote system so eliminating some extra manual steps. As a last preparatory step, the scalability workflow model is imported into the project and a new runtime configuration for the BT-MZ benchmark using class C input is created in the internal database using the provided by the framework GUIs.

Before proceeding with the configuration and automatic execution of the scalability workflow, the build process of the analyzed application has to be also adapted. This demands that each compiler invocation is manually prefixed with a special `$(PREP)` variable. This approach allows the framework to automatically instrument the source code using different tools if this is required by the analysis process. In the current study, the line specifying the MPI compiler in the main Makefile of NPB-MZ had to be appropriately extended.

As previously discussed, the scalability workflow starts by interactively asking a couple of questions about the runtime settings of the current experiment and then generates all sub-tasks depending on user's input. During this experiment, the workflow was configured to run the BT-MZ benchmark with up to 128 processors using different configurations of the number of MPI processes and OpenMP threads. As a performance monitoring tool, the Score-P measurement library was selected.

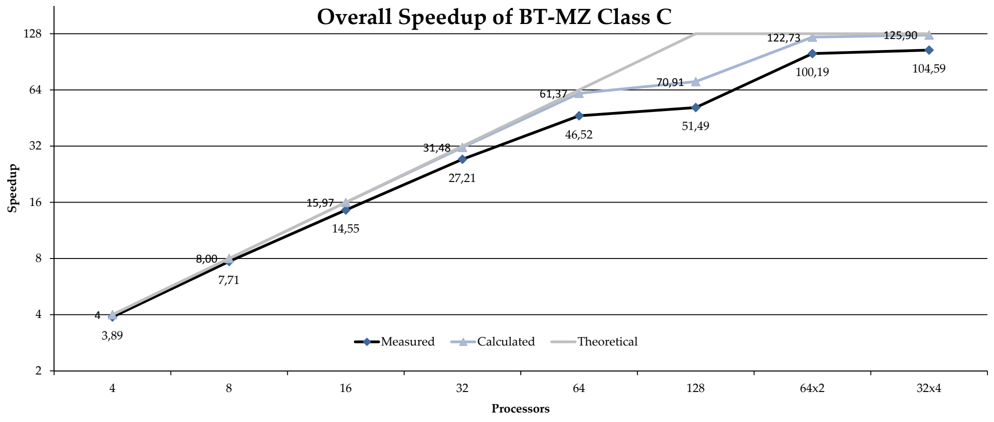
After collecting all required information, PAThWay created a snapshot of the current state of the NPB-MZ source code and synchronized that with the version available on the MPP segment of the linux cluster. This way, the local changes to the build process and other related files were pushed to the target machine and the NPB-MZ suite was prepared for automatic instrumentation and analysis.

After successfully completing these tasks, the framework generated separate batch scripts for all sub-experiments, instrumented the BT-MZ benchmark using Score-P as requested and finally submitted the jobs to the SLURM scheduler of the MPP cluster. The framework then entered monitoring mode and started waiting for the parallel jobs to complete. Whenever an experiment started, the environment detection tool recorded internally the provided running environment and then the BT-MZ code was executed with the specified number of processors. Upon completion of every job, PAThWay created a database entry for it including meta-information such as standard output and error, location of the result files, used external modules and others.

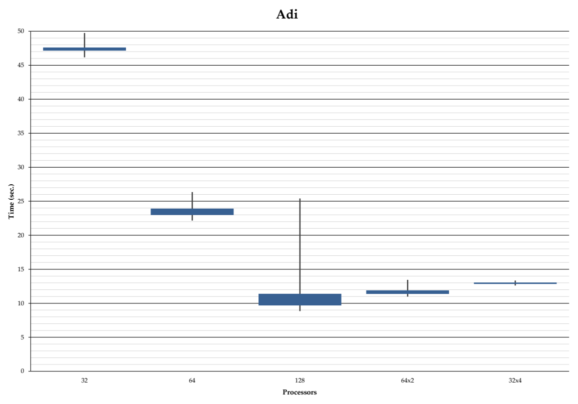
Since the MPP segment of the linux cluster of LRZ was heavily utilized during this experiment, the workflow execution was paused after the results from the first 4 sub-tasks were successfully recorded. At that point, the framework persisted locally the state of the workflow process and the Eclipse IDE was completely closed. On the next day, the workflow was restarted using the saved image. It automatically entered monitoring mode and checked the status of all previously pending batch jobs. Since all of them were completed by that time, PAThWay collected the experiment results, created the accompanying database entries, and showed a notification that the whole scalability study was successfully completed.

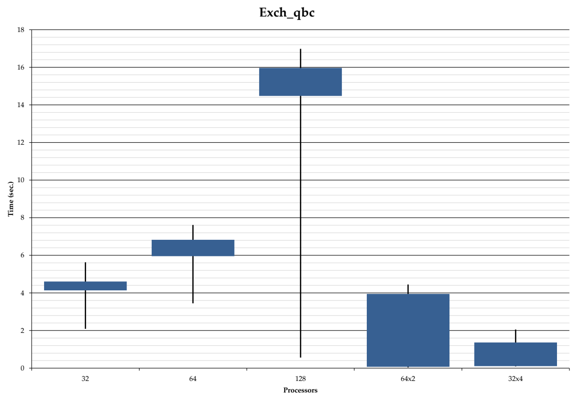
## 12.6. Results Exploration

After application monitoring, the next step in every performance engineering study is the analysis of collected runtime data. The very first task deals with extracting performance metrics such as



(a) Overall Scalability





(b) Computation (c) Communication

Figure 12.2.: Scalability of BT-MZ from NPB Multi-Zone

execution time, memory consumption, hardware counters, etc. Then, one must aggregate these metrics in order to be able to thoroughly comprehend the execution behavior of the analyzed application and plan any further analysis and tuning steps.

During the configuration of the current workflow process, the Score-P measurement library was chosen as a performance engineering tool. It was set to perform a profile analysis without measuring any hardware counters or collecting tracing data. Among all performance data storage formats supported by Score-P, the Cube 4 [189] format was selected. Using the utilities provided with the Cube toolkit, it was straightforward to extract the required performance metrics from the collected data for each code region of BT-MZ. Furthermore, the distribution of the metrics and their variance among the processes was calculated and used to create a fine-grained scalability model of the benchmark.

Table 12.2.: Scalability Results of NPB BT-MZ Class C

| | | Number of Processes and Threads ($N_p$ x $N_t$) | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Metric | Region | 1x1 | 16x1 | 32x1 | 64x1 | 128x1 | 64x2 | 32x4 |
| Time (sec.) | Main | 1426.91 | 98.10 | 52.45 | 30.67 | 27.71 | 14.24 | 13.64 |
| | ADI | 1413.17 | 94.15 | 47.53 | 23.69 | 11.83 | 11.83 | 12.96 |
| | Exch_QBC | 7.53 | 4.78 | 5.63 | 7.62 | 16.98 | 1.85 | 0.51 |
| | MPI_Waitall | - | 2.72 | 3.91 | 5.90 | 13.88 | 1.73 | 0.36 |
| | OMP | 1417.98 | 94.60 | 47.73 | 23.78 | 11.86 | 11.88 | 13.05 |
| Speedup | Main | 1.00 | 14.55 | 27.21 | 46.52 | 51.49 | 100.19 | 104.59 |
| | ADI | 1.00 | 15.01 | 29.73 | 59.64 | 119.46 | 119.48 | 109.03 |
| | Exch_QBC | 1.00 | 1.57 | 1.34 | 0.99 | 0.44 | 4.07 | 14.67 |
| | MPI_Waitall | - | 2.17 | 1.51 | 1.00 | 0.43 | 3.41 | 16.33 |
| | OMP | 1.00 | 14.99 | 29.71 | 59.64 | 119.58 | 119.38 | 108.65 |
| Eff. | Main | 100.00% | 90.91% | 85.02% | 72.68% | 40.23% | 78.28% | 81.71% |

After analyzing the extracted data, it was found that the instrumented version of the BT-MZ benchmark using Class C input data has a close to linear speedup with up to 64 MPI processes (see Figure 12.2a). In order to better understand that behavior, the distribution of the execution time among the processes for the two main phases of the benchmark was compared in Figures 12.2b and 12.2c. These charts show the minimum and maximum execution times for functions `Adi` and `exch_qbc` together with the 25% and 75% quantiles of the time distribution. As it can be seen, there is a big increase in the variance of the execution time metric for the `Adi` function when going from 64 MPI processes to 128. Since this function is responsible for the numerical solution of the Navier-Stokes equations, this signals for the appearance for a load imbalance condition. This state can be further confirmed by the huge variability of the communication time incurred by the exchange of the locally computed boundary values in function `exch_qbc`.

These results can be explained by the way BT-MZ distributes the discretization zones among the processors: the MPI processes with lower numbers get bigger chunks of the computational domain and thus take much longer to reach the numerical solution of the problem. This eventually results in longer waiting times during the communication phase. As it can be seen in Table 12.2, these large delays are induced by a MPI_Waitall code region and have direct effect on the scalability of the benchmark code and its efficiency in using the full power of the computational resources. The results also show that adding additional OpenMP threads to the execution of the BT-MZ instead of increasing the number of MPI processes results in better load balancing of the code. This is due to the fact that the benchmark automatically assigns more threads to the processes with larger zones so that they can reach a solution faster. As a result, all processes spend approximately equal time in the computation phase. This eventually leads to shorter waiting times during the exchange of boundary values and so to an improvement in the scalability and in the efficiency of the BT-MZ benchmark.

Chapter 13

# Cross-Platform Memory Analysis Workflow

**Contents**

**13.1. Cross-Platform Memory Analysis Process** . . . . . . . . . . . . . . . . . . . . . . . 153
**13.2. Workflow Model** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154
**13.3. STREAM Memory Benchmark** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 155
**13.4. Runtime Environment** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 156
**13.5. Generic Memory Analysis Strategy for Periscope** . . . . . . . . . . . . . . . . . . . 157
**13.6. Workflow Execution** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 158
**13.7. Results Exploration** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 159

## 13.1. Cross-Platform Memory Analysis Process

The precise analysis and comparison of different HPC systems is an important performance engineering process as it is fundamental for choosing the most suitable architecture for a specific parallel application (see Chapter 6.3). It can include multiple measurements of different components of HPC machines using both synthetic benchmarks and small computational kernels that are commonly extracted from user's applications. Among the normally collected performance metrics are the execution time, the number of floating-point instructions, the inter-process communication traffic and others.

However, due to the fact that since more than already 10 years processors have become much faster that computer's memory systems [225], often the performance of parallel applications is limited because of their memory operations. Factors such as the total aggregated memory bandwidth and the related access latencies have a stronger impact than the computational performance of processors. For example, simulation codes that do not use effectively the cache hierarchies of CPUs spend most of their execution time waiting for data to be updated after cache miss events.

Because of the crucial role a memory system has for the execution of today's HPC codes, a specialized cross-platform memory analysis workflow was developed within the PAThWay framework. Its main aim is to automate the analysis and comparison of different machines with respect to their memory characteristics. Furthermore, a new generic memory analysis strategy for the Periscope performance engineering toolkit was implemented and integrated into the workflow. Its goal is to automate the analysis of the runtime behavior of memory-related operations in highly parallel applications and thus help users to choose the most suitable HPC system for their particular use-case.

## 13.2. Workflow Model

Due to the popularity of the cross-platform performance engineering process, PAThWay provides a predefined workflow model for it as part of the whole framework. As it can be seen in Figure 13.1, this workflow starts by interactively asking the user to specify the required number of MPI processes and OpenMP threads. Depending on the underlying target of the analysis process, the user can then decide whether a specific performance analysis tool should be applied during the study or not. For example, a synthetic benchmark that stresses the memory system and, as a result, provides statistics about its performance might be exactly what one needs. In this case, users will typically run the uninstrumented version of that code without using any additional performance monitoring tools. However, in other circumstances when a comparison of the memory behavior of a specific real-world application is required, the usage of such a runtime measurement tool is a fundamental part of the process.

Following these two interactive configuration steps, the workflow model inquires about all HPC systems that will have to be used within the current cross-platform memory analysis. Once all machines are selected by the user, the process model will request the specification of an application to be executed on each of the systems and their exact configuration. Consequently, the corresponding performance engineering experiments will be configured according to all previously entered user requirements and sent to the underlying execution and monitoring system of PAThWay. Similar to the scalability analysis workflow, these will then be autonomously executed on each of the chosen HPC system. Finally, the results of all these experiments together with information about their precise runtime environments, standard output and error logs, execution statuses, etc., will be stored in the internal database of the framework so that they can be analyzed by the performance engineers later on.

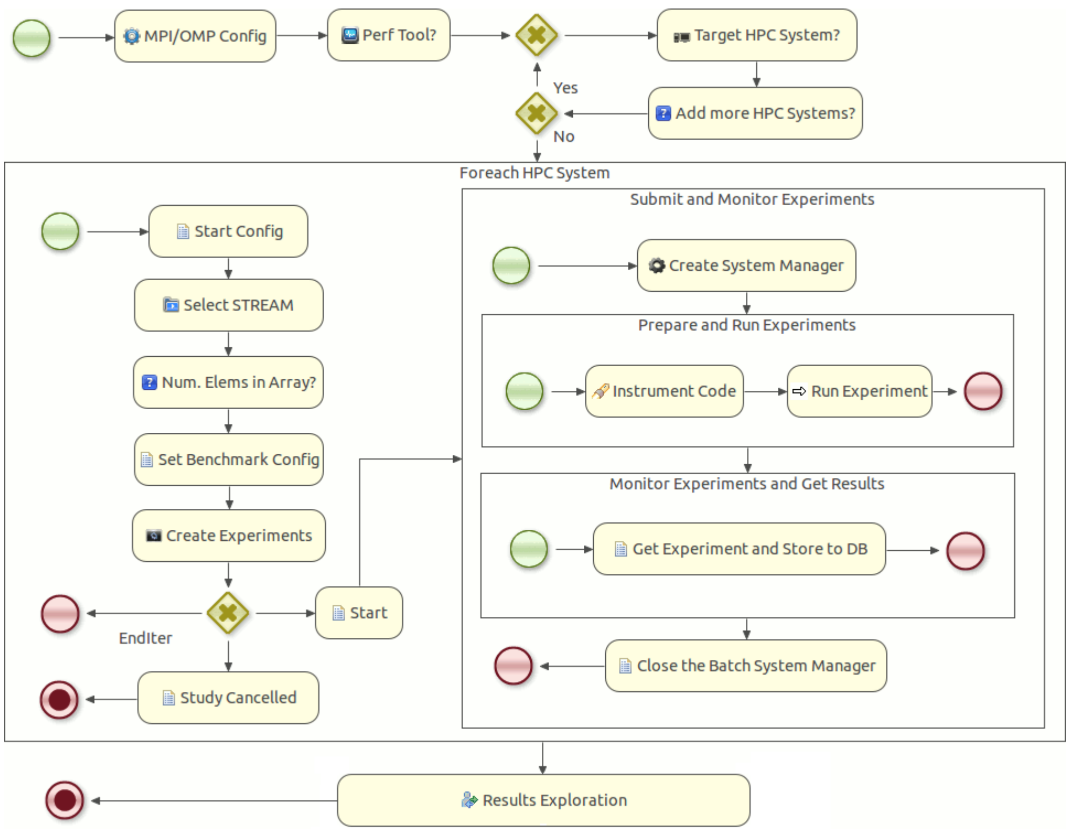


Figure 13.1.: Cross-platform memory analysis workflow model

## 13.3. STREAM Memory Benchmark

STREAM [144] is a very popular HPC benchmark that aims at measuring the sustained memory bandwidth and the computational rate of a machine. It was initially written in FORTRAN but was later on ported to the C programming language. It is parallelized using only OpenMP as it measures the memory performance within a single node and thus no intra-node communication is required. For the analysis of multiple-node setups, an MPI fork of the code is also available. However, this version is not very useful as it is simply running the same algorithm but on all nodes.

Internally, the benchmark uses four synthetic kernels with simple but very long vector operations which try to stress mainly the memory sub-system. The motivation behind this design is to create better performance estimations which account for the actual memory bandwidth of a system and not its theoretical maximum. Thus, the benchmark is more representative for real-world, memory-bound applications in comparison to the famous LINPACK [57] code.

Table 13.1.: STREAM benchmark: synthetic kernels [144]

| Kernel | | Bytes per | FLOPS per |
|---|---|---|---|
| Name | Operation | Iteration | Iteration |
| Copy | a(i) = b(i) | 16 | 0 |
| Scale | a(i) = scalar * b(i) | 16 | 1 |
| Sum | a(i) = b(i) + c(i) | 24 | 1 |
| Triad | a(i) = b(i) + scalar * c(i) | 24 | 2 |

As shown in Table 13.1, the different types of vector operations, which are performed by the STREAM benchmark, can be differentiated by the total number of floating point operations they include. However, the last two kernels, i.e. Sum and Triad, are also designed to stress other common components such as load-store memory ports and the existence of fused multiple-add operations.

## 13.4. Runtime Environment

Due to the nature of a cross-platform memory analysis workflow, multiple high-performance systems are needed for its successful execution. That is why, three of the leading HPC machines in Germany were selected: the MPP segment of the linux cluster at the Leibniz Supercomputing Centre (LRZ), LRZ's new petascale supercomputer called SuperMUC and the JuRoPA (Jülich Research on Petaflop Architectures) computational cluster from the Jülich Supercomputing Center. Whereas a detailed description of the first system is provided in Chapter 12.3, the other two will be shortly discussed in the following paragraphs.

SuperMUC, being the successor of LRZ's HLRB2 supercomputer, provides high-performance computing services for many research institutions and universities not only in the state of Bavaria but also throughout Europe. It was officially installed in June 2012 and since then it also serves as a Tier-0 center of the Partnership for Advanced Computing in Europe (PRACE) consortium. It has a peak performance of 3.185 Petaflop/s and consists of 9421 nodes separated in 18 thin partitions each having 512 compute cards and one fat-node island with 205 nodes. Each node in the thin partitions is composed of two *Intel Xeon E5-2680 8C* processors having a Sandy Bridge microarchitecture. These CPUs have 8 cores each and support 2-way hyperthreading which have a total peak performance of 21.6 GFlops per core. The memory bandwidth per processor is 51.2

GB/s and the latency to local memory is around 50 nanoseconds. The compute cards in the fat-node island have four Westmere-EX (*Intel Xeon E7-4870 10C*) processors each having 10 cores with 2-way hyperthreading. Each core has a peak performance of 9.6 GFlop/s, 70 nanoseconds local memory latency and a total of 34.1 GB/s memory bandwidth per processor.

The nodes within each partition of SuperMUC are interconnected via a non-blocking Infiniband switched fabric network produced by Mellanox Technologies. Four links each having a fourteen data transmission rate (FDR-10) are integrated together and used to connect the thin nodes. They have an effective theoretical data rate of 41.25 Gbit/s in each direction and up to 1 microseconds end-to-end MPI latency. The nodes within a fat partition are connected using quad data rate (QDR) Infiniband interconnect with an effective unidirectional rate of 8 Gbit/s.

The job scheduling on SuperMUC is managed from IBM's LoadLeveler [112] scheduler. Typical for LoadLeveler is that it always allocates computational resources for the jobs in chunks of complete nodes, i.e 40 cores for the fat-nodes and 16 for the thin-ones.

JuRoPA is the second biggest supercomputer in the Jülich Supercomputing Centre after the Blue Gene/Q system, which is in Jülich as well. It consists of two partitions: JSC and HPCFF. The first one consists of 2208 compute nodes which have an aggregated theoretical peak performance of 207 Teraflop/s whereas the HPCFF island has 1080 nodes in total and a maximum performance of 101 Teraflop/s. Each compute node of both partitions has 24 GB of operating memory and two quad-core Nehalem-based Intel Xeon X5570 processors. They support 2-way multiprocessing and have three levels of cache: 64KB L1 (32KB data and 32KB instruction cache) and 256KB L2 cache per core and 8MB L3 cache shared among the four cores. The peak performance of each core is 11.7 GFlop/s and the total memory bandwidth is 32 GB/s. JuRoPA uses a QDR Infiniband interconnect with a non-blocking Fat Tree topology. The underlying job scheduler of the cluster is Moab/TORQUE [2] which supports both batch and interactive parallel jobs with no node-sharing support.

## 13.5. Generic Memory Analysis Strategy for Periscope

As an attempt to narrow the gap between the performance of processors and memory systems, the architecture of every modern CPU includes a hierarchy of very fast cache modules. These are used to reduce the average data access time in computer applications by boosting the reuse of already loaded information. Effectively, this technique leads to lowering program's memory bandwidth requirements and thus decreasing its overall execution time. That is why, analyzing and optimizing the runtime behavior of programs with respect to their memory accesses plays a crucial role in every performance engineering project.

In order to allow flexible analysis of such operations within this workflow model, the Periscope performance engineering toolkit was extended to include a new search strategy for detecting generic memory bottlenecks. By automatically monitoring the execution of parallel applications and processing on-the-fly the low-level performance data collected from CPU's hardware counters, it is able to detect and report different memory-related performance properties.

This automatic search strategy uses a set of performance properties that encapsulate knowledge about common pitfalls of typical memory systems. As discussed in [22], these properties utilize internally the Performance Application Programming Interface (PAPI) [31] in order to abstract the access to the various cache-related hardware counters of a processor and define the high-level performance bottlenecks that describe inefficient usage of any of the three cache levels of modern CPUs. This approach allows Periscope to apply the very same analysis logic on multiple different HPC platforms without requiring a custom implementation for all of them.

## 13.6. Workflow Execution

The cross-platform memory analysis workflow was executed on three of Germany's top high-performance systems: the linux cluster at LRZ, their SuperMUC supercomputer and the JuRoPA cluster from the Jülich Supercomputing Center. Since the first two machines were also extensively used throughout the development of PAThWay, they are also natively supported by the framework and can be easily selected at process execution time. The third one, however, is not initially supported and thus required manual configuration in order to be used within the performance engineering workflow. That is why, the first step before starting with the cross-platform memory analysis dealt with adding the settings for connecting and using the JuRoPA system to PAThWay's internal storage. As described in Chapter 11.4, that required creating a new entry using the framework's custom GUIs. Furthermore, since the underlying job scheduling system of that machine, i.e. the Moab/TORQUE scheduler, was also not supported by PAThWay, an XML-based resource manager description had to be written for it and also registered in the internal database.

Once these preparatory steps were completed, an Eclipse project was created for the STREAM memory benchmark and its runtime configuration was also added to PAThWay's database. Similarly to the scalability analysis workflow, the build process of the project had to be also adapted by prefixing each compiler invocation in the corresponding *Makefile* with a special `$(PREP)` variable.

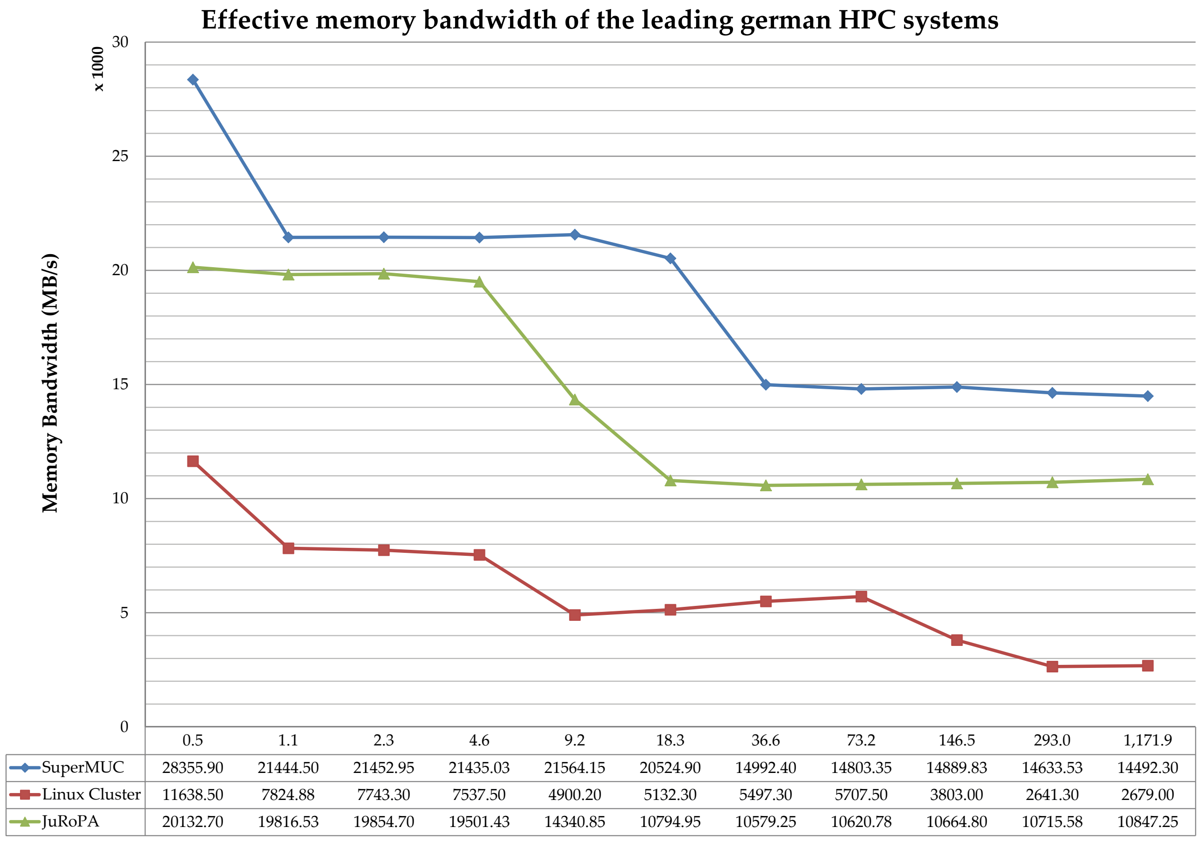
As the next step, the provided cross-platform workflow model was imported in an Eclipse project and it was started using the corresponding entry from PAThWay's menu. During the interactive configuration that comes with the process, the three target HPC systems were selected. The experiments were then configured to run with 1 MPI process and 1 OpenMP thread as they were targeting the performance of the memory system of only a single node and not that of the whole distributed system. Using this configuration, a study was performed using the STREAM benchmark without any extra code instrumentation. The idea was to measure the effective memory bandwidth of a node on each of the three HPC systems.

## 13.7. Results Exploration

Similarly to the previously discussed scalability analysis workflow, the next step after the automatic execution of the experiments is to analyze the generated performance data. Being specifically designed to investigate the memory capabilities of HPC systems, STREAM provides plenty of information about the runtime behavior on its own. That is why, the study, which was used to evaluate based on the presented cross-platform memory analysis workflow, utilized no external performance monitoring tools or libraries. Instead, the uninstrumented version of the STREAM benchmark was executed multiple times on the three HPC systems while varying sizes for the internal array structures. Using this approach, a performance engineer can easily explore the effects of the internal processor caches on application's memory performance and, thus, obtain a better approximation of the sustainable memory bandwidth. This value can then be used, for example, within performance engineering models of real-world applications in order to better estimate their runtime behavior on a given HPC system and decide whether the data access rates are the limiting factor for the performance.

As it can be seen in Figure 13.2, the performance engineering study was executed using 11 different sizes for STREAM's internal data storage arrays ranging from 500 KBytes up to 1172 MBytes. This way, the sustained bandwidth of both the L3 cache and the main memory system can be analyzed. These can be identified by the horizontal sections of the effective memory bandwidth graphs presented in Figure 13.2. For example, when using the array sizes between 1.1 MBytes and 18.3 MBytes, the data fits completely into the L3 cache of the *Intel Xeon E5-2680 8C* processors installed on the SuperMUC machine. This allows for a very fast data access with a bandwidth of up to 21 GBytes/sec. However, by doubling the STREAM's array sizes to 36.6 MBytes, the data does not fit anymore in the 20 MBytes L3 cache and has to be fetch from the main memory. This effectively results in a 7 GBytes/sec drop in the sustained memory bandwidth and corresponds to the maximum data access rate memory-bound applications can achieve.

By looking at the results, one can also mention that the runtime behavior of JuRoPA's memory system is also comparable to that of SuperMUC. The main differences are the smaller size of the L3 caches and the overall lower sustained main memory bandwidth of 10 GBytes/sec. At the same time, the performance behavior of memory-bound applications running on LRZ's Linux cluster



| | 0.5 | 1.1 | 2.3 | 4.6 | 9.2 | 18.3 | 36.6 | 73.2 | 146.5 | 293.0 | 1,171.9 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SuperMUC | 28355.90 | 21444.50 | 21452.95 | 21435.03 | 21564.15 | 20524.90 | 14992.40 | 14803.35 | 14889.83 | 14633.53 | 14492.30 |
| Linux Cluster | 11638.50 | 7824.88 | 7743.30 | 7537.50 | 4900.20 | 5132.30 | 5497.30 | 5707.50 | 3803.00 | 2641.30 | 2679.00 |
| JuRoPA | 20132.70 | 19816.53 | 19854.70 | 19501.43 | 14340.85 | 10794.95 | 10579.25 | 10620.78 | 10664.80 | 10715.58 | 10847.25 |

Figure 13.2.: Comparison of the effective memory bandwidth of SuperMUC, LRZ's linux cluster and JuRoPA high-performance systems

follows a bit different pattern. As shown by the lowest line in Figure 13.2, the maximum memory bandwidth is approximately two times slower that both the SuperMUC and JuRoPA systems. This can be explain by the fact that the Linux cluster utilizes the oldest processors of the three HPC machines having 2 x 6 MBytes L3 cache shared among all cores. Interestingly, however, is that another drop in the sustained memory performance can also be identified when the size of the STREAM's data structures is increased over 75 MBytes. After this point, the bandwidth will steadily fall until it reaches a constant rate of 2600 MBytes/sec for all data sizes above 300 MBytes. This can be cause by either fault memory modules or the fact that only one core is used throughout the experiment and the benchmark is not able to completely saturate the bandwidth because of processor's internal design. However, in order to identify the exact reason for this behavior, detailed investigation would be required which is out of the scope of this work.

Chapter 14

# Exhaustive Benchmarking Workflow

**Contents**

**14.1. Exhaustive Benchmarking Process** . . . . . . . . . . . . . . . . . . . . . . . . . . . . 161

**14.2. Workflow Model** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 162

**14.3. SPEC MPI-2007 Benchmarks** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 164

**14.4. Online Data Clustering Support for Periscope** . . . . . . . . . . . . . . . . . . . . . . 164

**14.5. Workflow Execution** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 166

## 14.1. Exhaustive Benchmarking Process

In this chapter, the exhaustive benchmarking performance engineering process will be introduced and its workflow model will be designed using PAThWay. Even though this process is not so well known by the normal HPC users, it is very often applied by hardware vendors and technical consultants. This is typically done as part of a standard pre-sales phase during which the machines of different HPC vendors and their runtime configurations are extensively tested according to specific client's requirements. In contrast to the cross-platform memory analysis workflow, this process measures the performance of all modules of a system, e.g. CPUs, memory systems, communication interconnects, file systems and others, either using small extracts of real-world applications which resemble the future usage of a system or by running directly the target simulation codes. Either way, this process requires the completion of multiple performance engineering experiments which makes is a good choice for automation using workflows.

Normally, such an exhaustive benchmarking process results in large amounts of performance data which have to be processed postmortem by the involved parties. Analyzing this information, however, can be a very tedious and error-prone work. That is why, the workflow model presented in this chapter interfaces with the Periscope performance engineering toolkit in order to enhance this phase. As discussed in Chapter 5.3.1, this tool is able to perform online analysis of the runtime behavior of applications and as such requires less post-processing of the results. What is more, after the implementation of an additional extension, Periscope is now also capable of autonomous clustering the gathered performance properties [166], thus further decreasing the required postmortem data analysis and improving the productivity of performance engineers.

## 14.2. Workflow Model

Provided with PAThWay, there is a simplified workflow model which tries to automate the exhaustive benchmarking process described in the previous section. Similarly to the other two process models, this one also consists of an interactive configuration phase and an automatic, back-end execution and monitoring of performance engineering experiments. However, due to the fact that an all-inclusive benchmarking process has to complete multiple different types of performance studies, its workflow model is more complicated and results in whole hierarchy of analysis experiments. That is why, one of the very first steps is to create a meta-experiment which will serve as the top-most parent of all performed experiments in order to create a well-organized structure of the collected data (see Figure 14.1). After that, the user is asked to choose the HPC system that has to be benchmarked, the required runtime settings for the parallel execution environments and the application that should be used throughout the process together with all required input data files and command-line arguments. This last step, specifically, consists of an iterative selection of the application entries, which are stored in the internal database of the framework, just like previously done for specifying all HPC system involved in the cross-platform workflow.

In order to simplify the workflow model for the sake of easier demonstration of its essential idea, it currently contains only a single MPI-based performance analysis step followed by a measurement of the single-node performance of the system. In reality, however, this model will also include other benchmarking steps targeting the memory system, the file system, the underlying communication network and other components of the analyzed HPC machine.

As it can be seen from the model in Figure 14.1, once the basic configuration is completed, the execution of workflow will be automatically diverted in one of the two parallel branches, i.e. MPI and HWC (hardware counters) ones. This is controlled by an internal variable specifying the current step of the study which is initially set to activate a performance analysis of the MPI communication behavior. Once the workflow enters this right branch of the model, it will select the Periscope performance engineering tool and will configure its analysis strategy. Furthermore, any additional environmental variables can be also specified in order to fine-tune the execution of the MPI library.

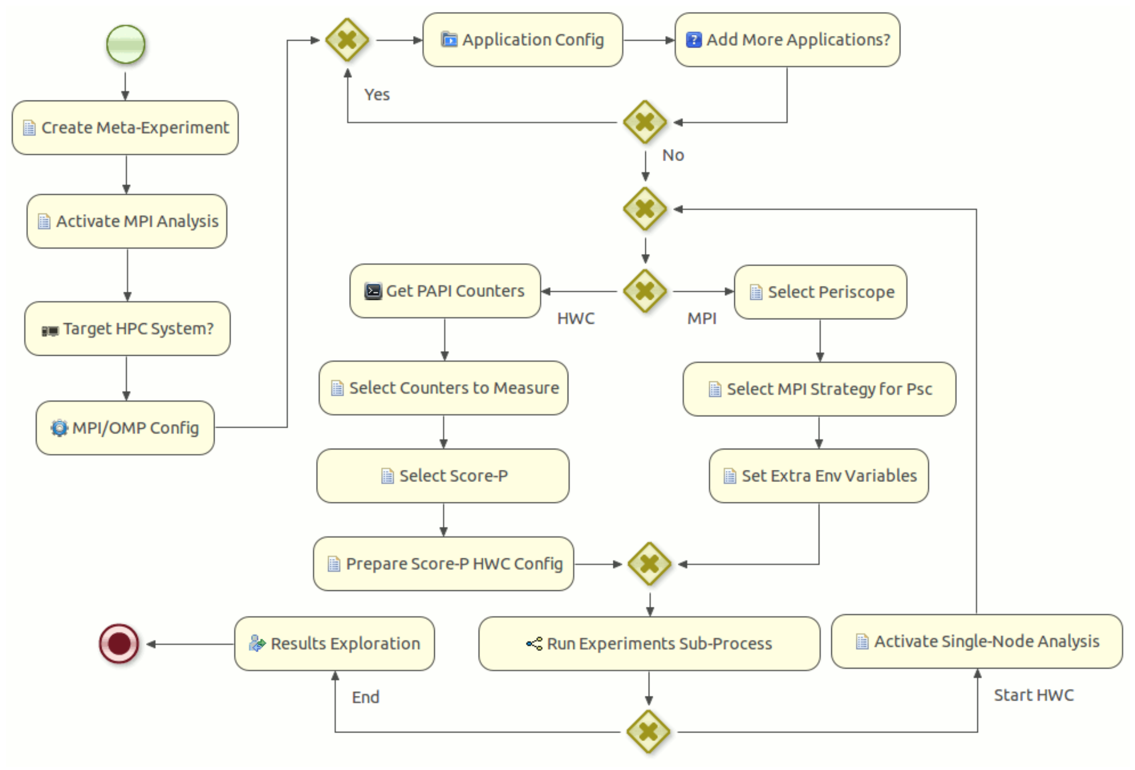


Figure 14.1.: Exhaustive benchmarking workflow model

Whenever the configuration of Periscope and its MPI analysis finishes, the workflow will generate all required performance analysis experiments and execute them on the target HPC system. In order to achieve better modularity of the exhaustive benchmarking workflow, this last phase is extracted into its own model which is embedded in the current workflow as an external sub-process.

After the MPI benchmarking experiments are completed, the workflow will return from the external sub-process and reach the exclusive split gateway shown at the lowest part of the model's layout. There the execution engine will have to decide whether to end the whole study and allow the user to explore the collected results or to prepare and execute another benchmarking step. In the current version of the process, the workflow will configure and execute one more benchmarking step after the MPI analysis completes. As mention previously, this last part will perform a single-node performance analysis using PAPI and the Score-P measurement libraries. Once this step is also completed, the workflow will finally terminate the execution and notify the user for the completion of the current exhaustive benchmarking study.

## 14.3. SPEC MPI-2007 Benchmarks

The Standard Performance Evaluation Corporation (SPEC) [192] is well-known, international consortium of multiple leading HPC hardware and software vendors, application developers and research institutions, the goal of which is to standardize the typical benchmarking process of HPC platforms and ease their comparison. As a result of the long-term collaboration between the participating organization, diverse performance measurement suites have been created to test most of today's HPC components including single-node performance, graphic cards, cluster interconnections, parallelization environments like MPI and OpenMP and others. Due to the broad coverage and extensive vendor support, the SPEC benchmarks are commonly used during the pre-sales benchmarking process and consecutive acceptance testing of new high-performance computing platforms.

One of the most popular benchmarking suites of the consortium is the SPEC MPI-2007 [152]. It targets specifically the MPI execution of compute-intensive, floating point applications on large HPC clusters which contain numerous symmetric multiprocessing (SMP) systems. In comparison to STREAM, this benchmark suite measures the interoperability of all components including CPU and memory architectures, communication interconnects and file systems and does not concentrate on a specific subset of them. As such, they do not provide very detailed analysis of the separate units but instead generate more realistic results that come closer to the performance of real-world applications.

The SPEC MPI-2007 suite includes a very large collection of benchmarks that cover a broad range of problems ranging from ocean and weather modeling to highly theoretical simulations like those from the field of quantum chromodynamics. Additionally, the suite provides two different input sizes for all codes: medium and large data sets. The first one is suitable for parallel execution using from 4 up to 128 MPI processes whereas the second one targets bigger runs requiring a minimum of 64 processes. Using the large inputs, some of the codes can scale up to 2048 MPI processes which makes them suitable for evaluating even larger HPC clusters.

## 14.4. Online Data Clustering Support for Periscope

Often, one of the biggest challenges that performance engineers face while investigating the runtime behavior of highly parallel applications is processing the tremendous amount of data that results from the analysis process. This quantity depends not only on the number of processors used throughout an experiment, but also on the underlying hardware and its monitoring counters, the derived performance metrics, application's runtime configuration and others. Even when using highly-optimized, online performance analysis techniques like those from the Periscope toolkit (see Chapter 5.3.1), the process still results in a large amount of very similar performance properties which are detected for the separate parallel processes.

In order to enhance the analysis phase, Periscope was successfully extended to include multivariate statistical clustering techniques which can effectively summarize similar properties and thus lower the amount of data that has to be processed. As discussed in [166], the Eclipse-based graphical user interface of Periscope was initially modified to support this functionality. Together with the Weka (Waikato Environment for Knowledge Analysis) [83] data-mining and machine learning framework, different clustering algorithms were tested on diverse performance data generated by Periscope in order to find the most suitable grouping criteria and technique. At the end, a separation of the properties based on their process distribution and the respective code region was chosen. As an underlying algorithm, the simple and well-established K-Means [109] method was selected.

In general, this clustering technique generates good quality clustering results when applied to Periscope's performance properties. However, in some circumstances, the resulting groups of similar bottlenecks were to fine-granular and close to each other. That is why, further research on the topic was done using more complex statistical clustering techniques together with preprocessing methods in order to achieve a faster and more accurate clustering output. For example, as discussed in details in [220] and [221], a modified version of a heuristic global optimization algorithm called harmony search [69] was used to select the optimal number of cluster centers and perform the grouping. Its results were then compared with those from the popular fuzzy c-means (FCM) [24] method. Furthermore, as a pre-processing step, individual outliers in the performance data were identified and filtered out using a Z-score [178] transformation method for data normalization.

Later on, some of these data clustering techniques were also integrated into Periscope's hierarchy of distributed analysis agents in order to achieve a dynamic, on-the-fly aggregation of the collected performance results. As a result, a more summarized version of the performance data collected from large-scale experiments could be generated by the tool without loosing valuable information. This way, performance engineers can more easily understand the complex nature of the results and even uncover unseen details about them.

## 14.5. Workflow Execution

Similarly to the scalability analysis workflow, the very first step in the preparation for starting a new exhaustive benchmarking process was to create an Eclipse project for the benchmark's source code. From the many different project types, the synchronized makefile project from PTP was again chosen as it provides very good support for remote development (see Chapter 5.5.2). Next, the build process of the SPEC MPI-2007 benchmark suite was adapted to the requirements of PAThWay. Those changes were then automatically synchronized with the MPP segment of the LRZ's linux cluster - the HPC system chosen for this particular performance engineering study.

At the end of this initial setup phase, the previously described workflow model was imported in a project and a new application configuration entry was created using PAThWay's corresponding graphical editor. From all benchmarks that come with the SPEC MPI-2007, the `127.wrf2` weather prediction code was selected due to its popularity among the HPC community and its overall good scalability with up to 128 processors. This test case implements the Weather Research and Forecasting (WRF) model, which is a mesocale numerical weather prediction algorithms designed for operational forecasting and atmospheric research [148]. It is a combination of both Fortran and C files and supports multi-level, fine-grained parallelism with both MPI and OpenMP processing. However, the version included with the SPEC MPI-2007 suite supports only distributed memory execution where the OpenMP mode is explicitly deactivated.

Once the source code was prepared and its runtime configuration was completed, the workflow was opened in the BPMN editor of Eclipse and it was started using PAThWay's execution menu. During the interactive setup phase, the Linux cluster at LRZ and the `127.wrf2` application were selected. The benchmarking study was set to use 128 MPI processes and only a single thread. After that, the workflow automatically entered the MPI analysis branch of the model and chose Periscope as a performance evaluation tool. Following, the framework started the *Run Experiments* sub-process, created a single experiment using the above runtime configuration, connected to the linux cluster and submitted the automatically generated batch job to the SLURM scheduler.

After a while, the submitted performance experiment for benchmarking the MPI communication of the system completed and the framework implicitly gathered the results and stored all the data in PAThWay's internal database. Afterwards, the last decision gateway was reached and since a single-node analysis is also explicitly requested in the model, the workflow instance started automatically to configure the second step of the benchmarking study. Internally, the framework connected to the cluster and fetched a list of all supported by the system hardware counters. This was done by running the `papi-decode` tool directly on the remote machine using a remote execution BPMN node. As a result, a comma-separated list with the available hardware counters was returned. This was then used to generate an interactive table view from where the following PAPI counters were selected for monitoring:

- PAPI_L3_TCA & PAPI_L3_TCM - total L3 cache accesses and misses;
- PAPI_FP_INS - total number of floating-point instruction
- PAPI_TOT_CYC - total CPU cycle required to execute the applicaiton.

These four hardware counters were specifically chosen as they are generally enough to get a first idea about the single-node performance of an application. It also has to be mentioned that all available hardware counters are often measured as part of such a exhaustive benchmarking study. However, due to limitations in the design of today's processors, they normally cannot be measured simultaneously and thus require many additional performance experiments. For the sake of simplicity, this behavior was not modeled in the current version of the workflow.

Following the selection of hardware counters to be monitored, the Score-P measurement library was specified as a performance engineering tool. Then, it was configured to observe the chosen 4 PAPI counters and report the value they obtain throughout the runtime of the WRF2 benchmark. Similarly to the MPI analysis step, once these configuration steps were completed, the external sub-process for execution of experiments was triggered. After successfully completing the single-node benchmarking task, the framework collected the runtime information about the experiment and created the corresponding database entry. At the end, a notification, saying that the workflow has finished, was shown. Finally, the collected performance results were manually analyzed and conclusions were made about both the single-core and the MPI performance of the Linux cluster at the Leibniz Supercomputing Center.

Chapter 15

# Generic Performance Tuning Workflow

**Contents**

**15.1. Performance Tuning and Its Application in HPC Centers . . . . . . . . . . . . . . 169**
**15.2. Generic Performance Tuning Workflow . . . . . . . . . . . . . . . . . . . . . . . . . 170**

## 15.1. Performance Tuning and Its Application in HPC Centers

In the last two decades, the field of high-performance computing experienced a vast development. From an area accessible only to a minority of highly technical computer specialists, it became adopted by a much broader public. Nowadays, it includes people from multiple science and engineering fields like biology, finance, physics and many others. These new types of HPC users possess a vast domain-specific knowledge about their areas but also have very diverse background in computer science ranging from basic to highly technical. As a result, the software they develop in order to solve their computational problems ranges from completely unoptimized sequential code to very scalable, hybrid programs that combine multiple parallel programming models.

This diversity leads to completely different execution profiles and requirements which influence the development of the whole HPC field. For example, power users are willing to invest a lot of time in applying new low-level programming models in order to squeeze the maximum performance out of today's highly parallel machines. However, the majority of HPC users are more interested in the numerical results delivered by their codes rather than the speed and the efficiency with which they are computed. This, combined with the idea that the computational resources are typically for free from user's point of view, leads to very inefficient usage of the machines and thus much higher operational expenses for the supercomputing centers [25]. That is why, for many of the HPC facilities nowadays it is more economically feasible to employ dedicated performance engineers rather than keep on buying newer and faster hardware. The task of these performance analysis and tuning experts is to try to proactively identify users with unoptimized codes which have the biggest impact on the under-utilization of the resources. After determining such cases, the performance engineers normally offer their tuning expertise for free to the involved users for the sake of optimizing their simulation codes and thus eventually lowering the overall operational expenses resulting from inefficient usage of the system.

## 15.2. Generic Performance Tuning Workflow

The previously discussed strategy of allocating dedicated performance engineers for identifying inefficient users' codes and successively optimizing their runtime behavior has proven to be a very suitable cost optimization measure for big HPC centers. The performance of multiple popular computational codes has been successfully improved, thus leading to better synergy between HPC users and supercomputing centers by not only lowering the typically huge budget for power consumption but also resulting in new scientific discoveries [107]. However, the process of performance tuning is still not well-established by the community and is often performed in a rather random approach depending only on the experience of the involved performance engineers. Consequently, its overall success can not be always guaranteed leading to higher costs for the HPC centers and dissatisfaction of the users.

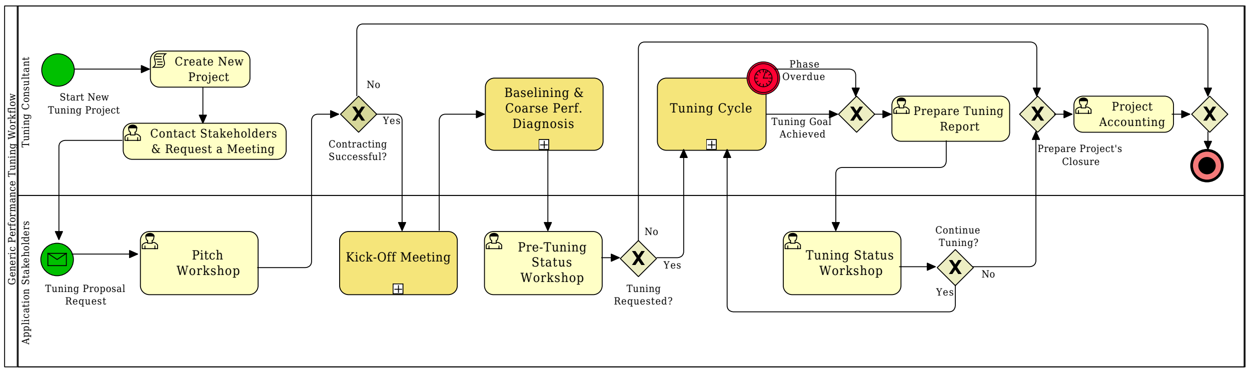


Figure 15.1.: Top-level overview of the performance tuning workflow

That is why, a specialized BPMN model is proposed within the framework of PAThWay with the goal of improving the tuning process and standardizing its execution. It builds on top of the performance analysis and tuning methodology presented in Chapter 1.2 and tries to provide a concrete implementation of high-level workflow proposed by Iwainsky et al. in [107]. As shown in the top-level overview of Figure 15.1, the process consists of multiple, interdependent steps and requires a close collaboration between the performance engineers and the HPC users. After identifying an inefficient application, the very first step of the tuning expert is to start off a new instance of the workflow and create a new project for storing related information. Then, he will need to contact the specific user and arrange for a pitch meeting to discuss the initial findings and the service that he provides. The goal of this event is to explain the need for optimizing user's code and serve as a motivation for the prospective future collaboration.

Once the user accepts the offer from the center's employee to optimize his code, the next step is to schedule an initial kick-off meeting. In the presented process model, this event is designed as an external BPMN sub-process in order to increase the implementation flexibility of the step and improve the modularity of overall workflow (see Figure 15.2). Since this is a real face-to-face session, it is modeled using multiple human tasks that aim to structure the meeting and collect all initial details about the tuning project. For example, it is crucial that a specific timeframe is explicitly defined as tuning activities can otherwise proceed forever. Moreover, all project's goals should be thoroughly discussed and well documented. That is why, this sub-process includes a loop over all of them which asks the performance engineer to enter goal's name, deadline and short description.

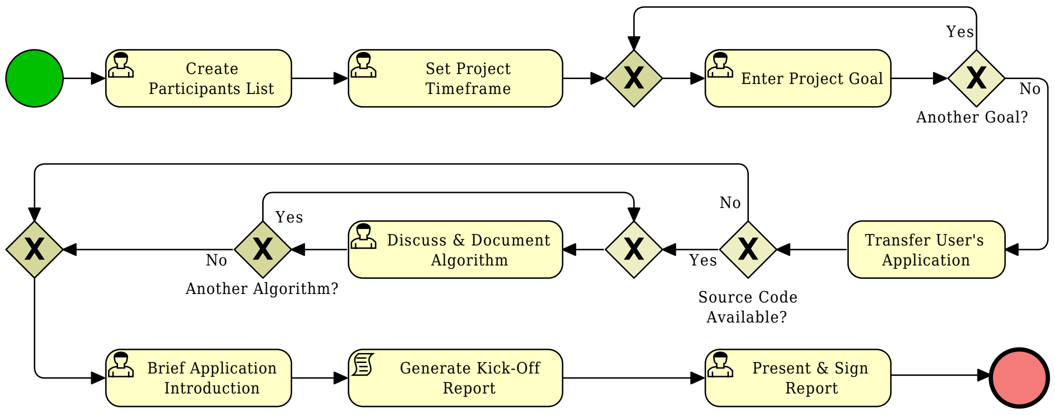


Figure 15.2.: Overview of the kick-off sub-process

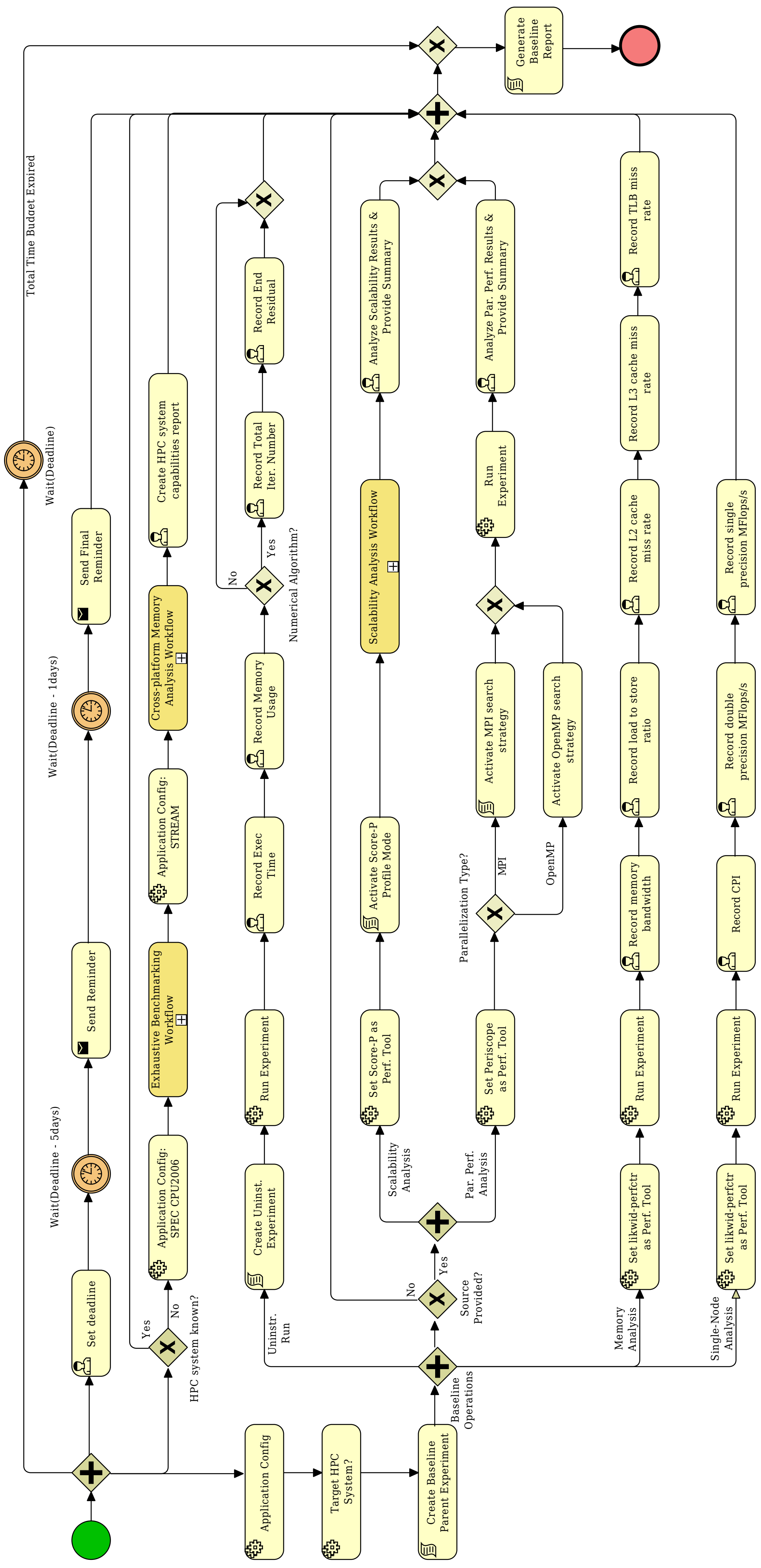
Another important step of this kick-off meeting is to agree upon the delivery of the user's application to the tuning expert. This can either include the full source code of the software or just the precompiled binaries. In the first case, this delivery should be additionally followed by a short discussion and documentation of the most important algorithms which, similarly to the specification of project's goals, is designed as a BPMN loop. Once the user's code or its binaries are successfully transferred to the performance engineer, a brief introduction to it must always be provided.

That is why, this step is also explicitly included in the workflow model of Figure 15.2. At the very end of the kick-off meeting, the process is designed to automatically generate a short report using PAThWay's documentation module containing summary of the event and all the data provided by the tuning expert such as defined goals, deadline, algorithms explanations and others. Finally, this report has to be once more presented to the user and officially accepted as a draft plan for the upcoming project steps.

Once the kick-off meeting completes and a short report with the initial project details is generated, the next step is to create a precise baseline for the future tuning activities and perform a high-level diagnosis of the runtime behavior of user's code. As it can be seen from the top-level process overview and Figure 15.3, this step of the performance tuning workflow is designed as a highly automated BPMN sub-process. It consists of a multitude of automatic performance analyzes targeting different aspects of the runtime behavior of the analyzed application. Furthermore, it includes a set of interactive user tasks for the initial configuration the baselining phase and the postmortem summarizing of the results.

The baseline step starts with the creation of a new configuration of the application in PAThWay's internal database and the selection of the target HPC system. In parallel, the workflow also asks the user to specify the final deadline for the phase and automatically activates a set of reminders and a process termination event in order to insure the baselining step will not continue forever and thus will stay in reasonable budget. In addition to these project management activities, the workflow will also ask the user whether he is already familiar with the target HPC machine. If this is not the case, the workflow will automatically start the previously discussed cross-platform memory analysis and exhaustive benchmarking workflows (see Chapters 13 and 14, respectively) in order to collect a variety of performance statistics about the underlying execution platform.

As Figure 15.3 shows, the next stage of the baselining process deals with the parallel execution of multiple performance analyzes on the user's code. These aim to automatically capture as much information about the runtime behavior of the application as required to create an overview of its performance problems without trying to identify their root causes. For example, one branch of the baselining operations runs an uninstrumented version of the code in order to record the overall execution time and memory usage. Additionally, in case of numerical codes, i.e. the most popular HPC application type, it also asks the performance engineer to specify a simple code-correctness criteria based on the number of iterations completed by the main algorithm and the final value of residual. All these data are internally recorded by the workflow and then are automatically included in the final baselining report.



Set deadline
Wait(Deadline - 5days)
Send Reminder
Wait(Deadline - 1days)
Send Final Reminder
Wait(Deadline)
Total Time Budget Expired
HPC system known?
Yes
No
Application Config: SPEC CPU2006
Exhaustive Benchmarking Workflow
Application Config: STREAM
Cross-platform Memory Analysis Workflow
Create HPC system capabilities report
Application Config
Target HPC System?
Create Baseline Parent Experiment
Baseline Operations
Uninstr. Run
Create Uninst. Experiment
Run Experiment
Record Exec Time
Record Memory Usage
Numerical Algorithm?
Yes
No
Record Total Iter. Number
Record End Residual
Source Provided?
Yes
No
Scalability Analysis
Set Score-P as Perf. Tool
Activate Score-P Profile Mode
Scalability Analysis Workflow
Analyze Scalability Results & Provide Summary
Par. Perf. Analysis
Set Periscope as Perf. Tool
Parallelization Type?
MPI
OpenMP
Activate MPI search strategy
Activate OpenMP search strategy
Run Experiment
Analyze Par. Perf. Results & Provide Summary
Memory Analysis
Set likwid-perfctr as Perf. Tool
Run Experiment
Record memory bandwidth
Record load to store ratio
Record L2 cache miss rate
Record L3 cache miss rate
Record TLB miss rate
Single-Node Analysis
Set likwid-perfctr as Perf. Tool
Run Experiment
Record CPI
Record double precision MFlops/s
Record single precision MFlops/s
Generate Baseline Report

Figure 15.3.: Overview of the baselining sub-process

In addition to the uninstrumented run, the workflow also automatically executes simple performance analyzes of the memory and the computational behavior of user's application. By interfacing with the LIKWID [204] performance analysis tool, it gathers statistics about the achieved memory bandwidth, cache miss-rates, the cycles per instruction (CPI) metric and the rate of single and double precision floating-point operations.

Furthermore, if the full source code of user's application was provided during the kick-off meeting, the workflow will also run two additional experiments to create profiles of the code's scalability and the performance with respect to the underlying parallel programming model. For example, a profile analysis of either the MPI or the OpenMP behavior will be performed using the Periscope toolkit. At the same time, the Score-P measurement library will be used throughout the sub-process for analyzing the scalability. Either way, at the end the workflow will interactively ask the performance engineer to provide a short summary about the results. Similarly to the data gathered during the other baselining operations, this information will be also automatically included in the final report based on PAThWay's documentation module.

After the project baseline is created, the next step in the performance tuning workflow is to arrange a pre-tuning meeting. The main idea of this event is to convey the findings of the baselining process and, together with the domain-specific knowledge of the user, identify potential performance problems. Then, after a detail discussion about the bottlenecks, a specific inefficiency should be selected out of them for tuning and a precise deadline for achieving that should be explicitly defined. If no concrete tuning agreement is reached with application's stakeholder, the performance engineer is asked to complete any necessary project accounting and officially terminate the current tuning process.

However, if an agreement on the set of performance bottlenecks to be fixed is successfully reached, the workflow enters the actual tuning cycle. Depending on the experience of the performance engineer, this process can proceed in many different ways and can have a varying impact on improving application's performance. As part of this workflow, however, a more structured tuning approach is proposed. As it can be seen in Figure 15.4, this cycle starts with the most simple but often very successful optimization technique - a search for the optimal set of compiler flags. The advantage of this approach is that it is quite simple as it does not require any complex source code modifications but instead relies on today's advanced compilers to do the hard work. At the same time, it can deliver the optimal configuration in a shorter time frame compared to any manual optimizations by simply interfacing with the Periscope performance engineering toolkit and its compiler flags tuning strategy.

Following this automatic search for optimal compiler flags is a manual task which compares the new runtime performance of the application with the previously generated baseline and the cur-

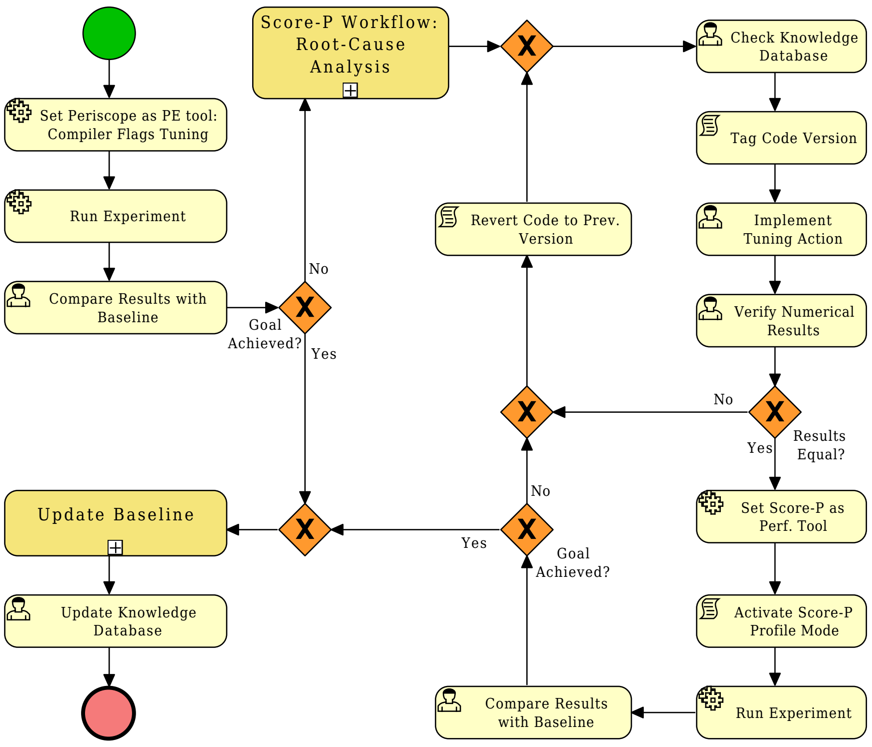

Score-P Workflow: Root-Cause Analysis
Check Knowledge Database
Set Periscope as PE tool: Compiler Flags Tuning
Tag Code Version
Run Experiment
Revert Code to Prev. Version
Implement Tuning Action
Compare Results with Baseline
No
Goal Achieved?
Yes
Verify Numerical Results
No
Results Equal?
Yes
No
Update Baseline
Yes
Goal Achieved?
Set Score-P as Perf. Tool
Update Knowledge Database
Activate Score-P Profile Mode
Compare Results with Baseline
Run Experiment

Figure 15.4.: Overview of the tuning cycle sub-process

rent tuning goals. If the last ones are successfully achieved, the current iteration of the tuning cycle completes. Then, the workflow automatically triggers an update the initial baseline configuration using the previously discussed baselining sub-process. Finally, an extended tuning report has to be manually prepared by the performance engineer and it has to be presented to the stakeholders during a special status meeting. Depending on the achieved results and user's expectations, another iteration of the tuning cycle can be consecutively initiated. Alternatively, the project can be terminated at this stage if any of the involved parties decides that further optimization efforts are not economically feasible.

In case the automatic search for compiler flags does not deliver the required performance boost or the user requests further optimizations during the status meeting, the workflow continues with a very detailed analysis of the root cause of the selected performance bottlenecks. This activity uses the Score-P measurement library to perform a sequence of profile and event tracing analyses as defined by the performance analysis workflow [213] of the Virtual Institute High Productivity Supercomputing (VI-HPS) [212].

After the source for performance slowdown is successfully identified, the user is requested to search for possible code modifications in order to tackle the detected problem. This can be ideally done using an internal database of previously encountered bottlenecks and their corresponding solutions. If the performance engineer has no access to such a knowledge collection within his organization, he is strongly advised to build on himself using, for example, PAThWay's documentation module. What is more, in order to further emphasize the importance of such a centralized tuning know-how, the tuning sub-process includes two extra BPMN nodes - one for asking the tuning expert to check the knowledge database for possible solutions and another one requesting him to document all performed modifications after every successful optimization iteration. This way, such a knowledge database can be easily built in the course of time without having to invest a lot of extra efforts.

The next step in the workflow, after identifying the root-causes of the performance problems and choosing suitable code modifications, is to create a snapshot of the current state of the code using PAThWay's internal revision control module (see Chapter 11.2). This task allows the tuning expert to always roll back any unsuccessful code modifications and eases the mapping between code versions and collected performance results. Following this activity is the usually long-running manual process of implementing the selected changes and the verification of the correctness of the resulting code. In case major deviations from the original behavior of the application are experienced, these last modifications have to be rolled back and new possible tuning ideas have to be identified. However, if the code-correctness criteria can be fulfilled, the workflow automatically executes a new profile-based analysis of the modified application and asks the expert to compare the newly collected results with project's baseline. At this stage, if the specified tuning goal is successfully achieved, the original baseline is updated and the performance engineer is requested to document the applied optimizations. However, if the runtime behavior of the new code version does not meet the acceptance criteria, the changes are rolled back and the search for new tuning ideas starts over again until either the tuning deadline is reached or a suitable optimization is identified and successfully implemented.

Finally, as shortly discussed in the previous paragraphs, a tuning report with all implemented code modifications, collected performance data and achieved results should be manually prepared by the tuning expert and presented to the stakeholders of the code. Following another intermediate status meeting, the user can then decide to request the optimization of further performance issues or to transfer back the newly optimized application and instruct the performance engineer to successfully close the project.

# Chapter 16

# Conclusion

**Contents**

**16.1. Summary** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **177**
**16.2. Future Work** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **180**
16.2.1. Interface with External Data Stores for Raw Performance Data . . . . . . . . 180
16.2.2. Automatic Post-Processing of Results and Process Reporting . . . . . . . . . 180
16.2.3. Performance Engineering Standardization and New Workflow Models . . . 181

## 16.1. Summary

This dissertation explored the concept of process automation in the field of parallel performance engineering and proposed a workflow-based modeling, execution and monitoring framework called PAThWay. It elaborated on the motivation behind this system which is to encapsulate the great complexity of computational environments, provide a more results-oriented analysis and tuning process and so decrease the time-to-delivery in HPC ventures. In order to achieve these goals, PAThWay incorporates ideas from the field of business intelligence in order to automatize and standardize the naturally long-running, iterative processes found in the area of high-performance computing such as scalability analysis of parallel applications, benchmarking of HPC systems, parameter studies and others. Furthermore, it also aims at enhancing the typically cyclic performance tuning process by providing means for tracking the source code evolution and integrating that within the overall software development life-cycle.

Due to the extremely fast evolution of the HPC field, the software architecture PAThWay was designed using three interconnected layers in order to be easily extendable without considerable efforts from the involved developers or the performance engineers. The topmost layer serves as the graphical user interface of the framework and is thus meant to ease the interaction with users. The second software layer includes multiple internal components which are responsible for processing the user-specified high-level workflows in order to generate and execute real activities out of them. For example, there are modules for managing the runtime configuration of experiments and applications, collecting and storing performance results and documenting the overall performance engineering process. Last but not least, the lowest layer of the PAThWay's architecture provides interfaces to different external components that contribute to the creation of the overall automation environment such as performance tools, a revision control system and the Parallel Tools Platform.

There are many different ways to automate iterative processes like the ones found in HPC-related projects. However, there is neither an established standard for doing this, nor a set of best-practices which are commonly found in other areas of industry and academia. In order to address these issues, PAThWay implements a powerful workflow modeling approach based on the broadly accepted standard for process modeling called Business Process Model and Notation (BPMN). Additionally, the framework provides a collection of domain-specific BPMN activities for adapting the standard to the typically inhomogeneous nature of the performance engineering field and thus enhancing the overall modeling and execution environment. These custom nodes serve not only as an interactive user interface during the execution of workflows but also enable the seamless integration of the rest of the underlying software components within the high-level process models. Furthermore, instead of trying to develop a custom workflow engine, the middle layer of the framework interfaces with the popular open-source jBPM/Drools suite.

The second major component in the middle layer of PAThWay's architecture is an internal data storage repository. It is designed to provide a flexible representation of the data required for the successful execution of workflows such as different configurations of user's applications, HPC systems, performance engineering tools and others. Additionally, it stores comprehensive information about the runtime environment of performance experiments, performance measurement results and even project documentation. Internally, this software component is based on an embedded version of the open-source H2 database engine so that the data scheme can be easily extended at any time in the future. Furthermore, following modern software engineering practices, all data manipulation operations were not implemented using pure SQL queries but rather with the help of an established, java-based object-relational mapping framework called Hibernate.

As discussed throughout this thesis, having a central and easy to use documentation describing requirements, work progress and project goals is another crucial criteria for the success of every project. Even though there are plenty of software tools that try to structure this information, most of them are rather complex to use as they are targeting the management of large software projects. That is why, they are not very popular among the HPC community where such data is often spread across multiple systems and performance experiments. Consequently, another software component was implemented as part of the middle layer of PAThWay in order to fill this gap and provide an integrated project documentation functionality. It is based on an customized version of the EclipseWiki editor for Eclipse and uses both Wiki-formatted text files and the internal database as back-end storage.

The bottom layer of the architecture of PAThWay consists of a set of supportive modules. These include a Git-based revision control system for tracking the code evolution during the typical performance tuning cycle, an environment detection tool for capturing the runtime configuration of experiments and an underlying execution and monitoring system based on the Parallel Tools Platform. In contrast to the previous software components, these modules are not directly visible to the users. Instead, they are only interfaced through the other components in order to build a more flexible and easily adaptable automation framework.

As part of the evaluation of PAThWay, this dissertation presented three popular performance engineering workflows that were successfully implemented within the framework. Firstly, a model for performing automatic scalability analyses of parallel applications was designed and then extensively tested using the NAS Parallel Benchmarks Multi-Zone suite. Secondly, a performance engineering workflow for evaluating and comparing the runtime performance of memory systems was discussed. In order to show its feasibility, the model was then executed on three different HPC systems using the STREAM memory benchmark and the results were again shortly discussed. Thirdly, the standard benchmarking processes that is often applied in supercomputing centers throughout a pre-sales phase is introduced and subsequently evaluated using the well-established SPEC MPI-2007 benchmark suite.

What is more, this thesis proposed a complete BPMN model for semi-automatizing the very common performance tuning workflow. It discussed the importance of standardizing this process from the point of view of HPC centers and the positive effect that this would have on their daily operations. Additionally, a simple but effective methodology for building a collection of tuning recipes was introduced as part of the workflow. The ideas behind this workflow model aim at laying the foundations for standardizing the way performance tuning is performed and at providing a real-world example on how HPC processes can be easily enhanced by adopting BPMN-based automation techniques.

Last but not least, this dissertation also discussed two new extensions to the Periscope performance engineering tool that were performed in order to create a more scalable, cross-platform analysis environment. Firstly, a portable memory analysis strategy was introduced and the motivation behind its implementation was briefly discussed. Secondly, different statistical clustering techniques for summarizing large-scale performance data were presented and their suitability to the given problem was examined.

## 16.2. Future Work

This dissertation laid the foundation for process automation in the field of performance engineering and proposed a prototype infrastructure for its realization using PAThWay. Consequently, these results also opened new research questions and motivated further implementation work. Among the many possible directions, three of the most important ones according to the author will be discussed in the following sections.

### 16.2.1. Interface with External Data Stores for Raw Performance Data

PAThWay provides a flexible framework for designing, executing and monitoring performance engineering workflows. Additionally, it is able to collect plenty of meta-information about the performed experiments such as runtime environment, execution logs and others. However, when it comes to organizing the raw performance data collected from these experiments, the framework has a very basic support. Even though it can store results from the Periscope toolkit directly into its database, data generated by other tools is neither processed nor internally stored. Instead, just a reference to the location of the collected information is saved within each experiment.

There are, however, other frameworks like TAUdb (previously called PerfDMF) [88] which are highly specialized in post-processing and management of large amounts of raw performance data. That is why, instead of extending PAThWay to perform such operations, these tools can be directly interfaced within the performance engineering workflows. For example, this way users will be able to perform cross-experiment comparisons or arbitrary manipulate the collected data in order to gain further insight about the runtime characteristics of their applications.

### 16.2.2. Automatic Post-Processing of Results and Process Reporting

Currently, only the configuration, execution and runtime monitoring of experiments is automated within PAThWay. The commonly following phase of results analysis is modeled as a manual BPMN activity which asks the user to explore the data and draw conclusions out of it on his own. This is, however, not a very productive approach and, to a certain extend, can also be easily automated. For example, after running all experiments in a scalability analysis workflow, the framework can directly process the collected results and extract key performance metrics out of it.

Following, it can generate scalability diagrams or even create analysis reports out of the data on behalf of the user.

Furthermore, the framework can be extended to include an activity monitoring component which will be responsible for tracking and reporting the usage of PAThWay's workflows. This information can be used within project reports or simply as a reference of the completed work.

### 16.2.3. Performance Engineering Standardization and New Workflow Models

Due to the early stage of PAThWay, only a very small set of performance engineering workflows were modeled using the framework. However, there are many other activities in the HPC field which can directly benefit from automation. That is why, to close up this list of future activities, the creation of further workflows is shortly discussed. It is believed that this will have the biggest impact on the future adoption of PAThWay as a workflow formalization tool and will trigger new discussions within community for the sake of standardizing the performance engineering field.

For example, a workflow implementing a very detailed and structured analysis of performance bottlenecks can be created. Its aim will be to define the best practices for performance analysis by standardizing on a set of performance engineering tools to be used together with the precise sequence in which they should be applied by the users. This will not only provide clear guidelines for novice performance engineers, but will also enhance the work of experienced ones by adding more structure to their daily operations. What is more, having easy to follow suggestions on how exactly analysis techniques should be applied can also improve the adoption of performance engineering tools by a broader HPC community.

Going one step further, PAThWay can also be extended to support the execution of complete, multi-step performance engineering projects. For example, the generic workflow for performance tuning presented in Chapter 15 can be applied to real-world optimization projects as those performed by the supercomputing centers. As a result, the quality and the success rate of performance tuning projects can be enhanced which will inevitably lead to better, more efficient operations for the centers and an increased overall user's satisfaction.

# Appendices

Appendix A

# Glossary

| | |
|---|---|
| API | Application Programming Interface |
| ARIS | Architecture of Integrated Information Systems |
| ATLAS | Automatically Tuned Linear Algebra Software |
| BAM | Business Activity Monitoring |
| BDE | Bottleneck Detection Engine |
| BIRT | Business Intelligence and Reporting Tools |
| BLAS | Basic Linear Algebra Subroutines |
| BPEL | Business Process Execution Language |
| BPEL4People | Extension to the BPEL standard for integration of human process participants |
| BPEL4WS | Business Process Execution Language for Web Services |
| BPM | Business Process Management |
| BPMI | Business Process Management Initiative |
| BPMN | Business Process Model and Notation |
| BPMS | Business Process Management Suite |
| CEP | Complex Event Processing |

| | |
|---|---|
| CLOB | Character Large Object |
| ConOps | Concept of Operations Document |
| CPI | Cycles Per Instruction |
| CPL | Common Public License |
| CPU | Central Processing Unit |
| CUPTI | NVIDIA's CUDA Profiling Tools Interface |
| CVS | Concurrent Versions System |
| DAO | Data Access Object |
| DVCS | Distributed Version Control System |
| EAI | Enterprise Application Integration |
| EPC | Event-driven Process Chains |
| EPML | EPC Markup Language |
| ERD | Entity-Relationship Diagram |
| ETFw | PTP External Tools Framework |
| FFT | Fastest Fourier Transform |
| FFTW | Fastest Fourier Transform in the West |
| FLOSS | Free and Open Source Software |
| FTP | File Transfer Protocol |
| GPGPU | General-Purpose Graphics Processing Unit |
| GT | Globus Grid Toolkit |
| GUI | Graphical User Interface |
| HMPP | Hybrid Multicore Parallel Programming |
| HPC | High Performance Computing |
| HPCS | High Productivity Computing Systems Toolkit |

| | |
|---|---|
| HR | Human Resources |
| HSQLDB | HyperSQL DataBase |
| HTTP | Hypertext Transfer Protocol |
| IDE | Integrated Development Environment |
| IEEE | Institute of Electrical and Electronics Engineers |
| IPC | Instructions per Cycle |
| ITIL | Information Technology Infrastructure Library |
| JAXB | Java Architecture for XML Binding |
| JDBC | Java Database Connectivity |
| JPA | Java Persistence API |
| JSON | JavaScript Object Notation |
| JVM | Java Virtual Machine |
| JVMPI | Java Virtual Machine Profiling Interface |
| LRZ | Leibniz Supercomputing Center |
| MathML | Mathematical Markup Language |
| MIASE | Minimum Information About a Simulation Experiment |
| MKL | Intel's Math Kernel Library |
| ML | Machine learning |
| MoML | Modeling Markup Language |
| MPI | Message Passing Interface |
| MRI | Monitoring Request Interface |
| MVEL | Java-based expression language |
| MWE | Modeling Workflow Engine from the Eclipse IDE |
| NCEAS | National Center for Ecological Analysis and Synthesis |

| | |
|---|---|
| OASIS | Organization for the Advancement of Structured Information Standards |
| ODBC | Open Database Connectivity |
| OEDL | OMF Experiment Description Language |
| OMG | Object Management Group Standardization Consortium |
| OOP | Object-Oriented Programming |
| OPARI2 | OpenMP Pragma and Region Instrumentor (version 2) |
| OpenCL | Open Computing Language |
| OpenMP | Open Multiprocessing |
| ORM | Object-relational mapping |
| OTF | Open Trace Format |
| OTF2 | Open Trace Format (version 2) |
| PAPI | Performance Application Programming Interface |
| PDT | Program Database Toolkit |
| PerfDMF | Performance Data Management Framework |
| PLDT | PTP Parallel Language Development Tools |
| PMPI | MPI Profiling Interface |
| PTP | Parallel Tools Platform |
| RDBMS | Relational Database Management System |
| RLS | Globus Replica Location Service |
| SBML | Systems Biology Markup Language |
| SCUFL2 | Simple Conceptual Unified Flow Language (version 2) |
| SDE | Solution Determination Engine |
| SDLC | Software Development Life Cycle |

| | |
|---|---|
| SED-ML | Simulation Experiment Description Markup Language |
| SIE | Solution Implementation Engine |
| SIR | Standardized Intermediate Representation |
| SMP | Symmetric Multiprocessing |
| SPEC | Standard Performance Evaluation Corporation |
| SQL | Structured Query Language |
| SRB | Storage Resource Broker |
| SRS | Software Requirements Specification |
| SSH | Secure Shell Protocol |
| SVN | Subversion Revision Control System |
| SyRS | System Requirements Specification |
| TAU | Tuning and Analysis Utilities |
| TDD | Test-Driven Development |
| TLB | Translation Lookaside Buffer |
| UI | User Interface |
| UML | Unified Modeling Language |
| URI | Uniform Resource Identifier |
| UUID | Universally Unique Identifier |
| VCS | Version Control System |
| VDL | Virtual Data Language |
| VI-HPS | Virtual Institute - High Productivity Supercomputing |
| WfMC | Workflow Management Coalition |
| WFMS | Workflow Management System |
| WS-BPEL | Web Services Business Process Execution Language |

| | |
|---|---|
| WSDL | Web Services Description Language |
| XML | Extensible Markup Language |
| XPDL | XML Process Definition Language |

Appendix B

# Custom Domain-Specific BPMN Nodes: XML Definitions

## B.1. Application Configuration

```
<task id=”25” name=”Application Config” tns:taskName=”AppConfig”>
 <ioSpecification>
     <inputSet></inputSet>
     <dataOutput id=”ApplicationOutput” name=”Application”/>
     <outputSet>
          <dataOutputRefs>ApplicationOutput</dataOutputRefs>
     </outputSet>
 </ioSpecification>
 <dataOutputAssociation>
     <sourceRef>ApplicationOutput</sourceRef>
     <targetRef>applVar</targetRef>
 </dataOutputAssociation>
</task>
```

## B.2. MPI and OpenMP Configuration

```
<task id=”10” name=”MPI/OMP Config” tns:taskName=”ProcThreadsConfig”>
 <ioSpecification>
    <inputSet></inputSet>
    <dataOutput id=”OMPThreadsOutput” name=”OMPThreads” />
    <dataOutput id=”MPIProcsOutput” name=”MPIProcs” />
    <outputSet>
        <dataOutputRefs>OMPThreadsOutput</dataOutputRefs>
        <dataOutputRefs>MPIProcsOutput</dataOutputRefs>
   </outputSet>
 </ioSpecification>
 <dataOutputAssociation>
    <sourceRef>OMPThreadsOutput</sourceRef>
    <targetRef>ompThreadsVar</targetRef>
 </dataOutputAssociation>
 <dataOutputAssociation>
    <sourceRef>MPIProcsOutput</sourceRef>
    <targetRef>mpiProcsVar</targetRef>
 </dataOutputAssociation>
</task>
```

## B.3. Target HPC System Selection

```
<task id=”4” name=”Target HPC System?” tns:taskName=”SelectTargetSystem” >
 <ioSpecification>
    <inputSet></inputSet>
    <dataOutput id=”HPCSystemOutput” name=”HPCSystem” />
    <outputSet>
        <dataOutputRefs>HPCSystemOutput</dataOutputRefs>
    </outputSet>
 </ioSpecification>
 <dataOutputAssociation>
    <sourceRef>HPCSystemOutput</sourceRef>
    <targetRef>hpcSystemVar</targetRef>
 </dataOutputAssociation>
</task>
```

## B.4. Performance Tool Selection

```
<task id="24" name="Perf Tool?" tns:taskName="SelectPerformanceTool" >
 <ioSpecification>
    <inputSet></inputSet>
    <dataOutput id="PerfToolOutput" name="PerfTool" />
    <outputSet>
       <dataOutputRefs>PerfToolOutput</dataOutputRefs>
    </outputSet>
 </ioSpecification>
 <dataOutputAssociation>
    <sourceRef>PerfToolOutput</sourceRef>
    <targetRef>perfToolVar</targetRef>
 </dataOutputAssociation>
</task>
```

## B.5. Source Code Instrumentation

```
<task id="5" name="Instrument Code" tns:taskName="InstrumentCode" >
 <ioSpecification>
    <dataInput id="BatchSysManagerInput" name="BatchSysManager" />
    <dataInput id="ExperimentInput" name="Experiment" />
    <inputSet>
       <dataInputRefs>BatchSysManagerInput</dataInputRefs>
       <dataInputRefs>ExperimentInput</dataInputRefs>
    </inputSet>
    <outputSet></outputSet>
 </ioSpecification>
 <dataInputAssociation>
    <sourceRef>batchSysManagerVar</sourceRef>
    <targetRef>BatchSysManagerInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>experimentVar</sourceRef>
    <targetRef>ExperimentInput</targetRef>
 </dataInputAssociation>
</task>
```

## B.6. Experiments Creation

```
<task id="6" name="Create Experiments" tns:taskName="CreateExperiments" >
 <ioSpecification>
    <dataInput id="PerfToolInput" name="PerfTool" />
    <dataInput id="OMPThreadsInput" name="OMPThreads" />
    <dataInput id="ApplicationInput" name="Application" />
    <dataInput id="MPIProcsInput" name="MPIProcs" />
    <dataInput id="HPCSystemInput" name="HPCSystem" />
    <dataInput id="ExpTypeInput" name="ExpType" />
    <dataOutput id="ExperimentsOutput" name="Experiments" />
    <dataOutput id="ConfigOKOutput" name="ConfigOK" />
    <inputSet>
        <dataInputRefs>PerfToolInput</dataInputRefs>
        <dataInputRefs>OMPThreadsInput</dataInputRefs>
        <dataInputRefs>ApplicationInput</dataInputRefs>
        <dataInputRefs>MPIProcsInput</dataInputRefs>
        <dataInputRefs>HPCSystemInput</dataInputRefs>
        <dataInputRefs>ExpTypeInput</dataInputRefs>
    </inputSet>
    <outputSet>
        <dataOutputRefs>ExperimentsOutput</dataOutputRefs>
        <dataOutputRefs>ConfigOKOutput</dataOutputRefs>
    </outputSet>
 </ioSpecification>
 <dataInputAssociation>
    <sourceRef>perfToolVar</sourceRef>
    <targetRef>PerfToolInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>ompThreadsVar</sourceRef>
    <targetRef>OMPThreadsInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>applVar</sourceRef>
    <targetRef>ApplicationInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>mpiProcsVar</sourceRef>
    <targetRef>MPIProcsInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>hpcSystemVar</sourceRef>
    <targetRef>HPCSystemInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>expTypeVar</sourceRef>
    <targetRef>ExpTypeInput</targetRef>
 </dataInputAssociation>
 <dataOutputAssociation>
    <sourceRef>ExperimentsOutput</sourceRef>
    <targetRef>experimentsListVar</targetRef>
 </dataOutputAssociation>
 <dataOutputAssociation>
    <sourceRef>ConfigOKOutput</sourceRef>
    <targetRef>configOkVar</targetRef>
 </dataOutputAssociation>
</task>
```

## B.7. Experiment Execution

```
<task id="8" name="Run Experiment" tns:taskName="RunExperiment" >
 <ioSpecification>
    <dataInput id="BatchSysManagerInput" name="BatchSysManager" />
    <dataInput id="ExperimentInput" name="Experiment" />
    <inputSet>
        <dataInputRefs>BatchSysManagerInput</dataInputRefs>
        <dataInputRefs>ExperimentInput</dataInputRefs>
    </inputSet>
    <outputSet> </outputSet>
 </ioSpecification>
 <dataInputAssociation>
    <sourceRef>batchSysManagerVar</sourceRef>
    <targetRef>BatchSysManagerInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>experimentVar</sourceRef>
    <targetRef>ExperimentInput</targetRef>
 </dataInputAssociation>
</task>
```

## B.8. Runtime Manager Creation

```
<task id="7" name="Create System Manager" tns:taskName="CreateBatchSystemManager" >
 <ioSpecification>
    <dataInput id="StartManagerInput" name="StartManager" />
    <dataInput id="ConnectionInput" name="Connection" />
    <dataInput id="ExperimentInput" name="Experiment" />
    <dataOutput id="BatchSysManagerOutput" name="BatchSysManager" />
    <dataOutput id="ConnectionOutput" name="Connection" />
    <inputSet>
        <dataInputRefs>StartManagerInput</dataInputRefs>
        <dataInputRefs>ConnectionInput</dataInputRefs>
        <dataInputRefs>ExperimentInput</dataInputRefs>
    </inputSet>
    <outputSet>
        <dataOutputRefs>BatchSysManagerOutput</dataOutputRefs>
        <dataOutputRefs>ConnectionOutput</dataOutputRefs>
    </outputSet>
 </ioSpecification>
 <dataInputAssociation>
    <sourceRef>startManagerVar</sourceRef>
    <targetRef>StartManagerInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>connectionVar</sourceRef>
    <targetRef>ConnectionInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>experimentVar</sourceRef>
    <targetRef>ExperimentInput</targetRef>
 </dataInputAssociation>
 <dataOutputAssociation>
    <sourceRef>BatchSysManagerOutput</sourceRef>
    <targetRef>batchSysManagerVar</targetRef>
 </dataOutputAssociation>
 <dataOutputAssociation>
    <sourceRef>ConnectionOutput</sourceRef>
    <targetRef>connectionVar</targetRef>
 </dataOutputAssociation>
</task>
```

## B.9. Load Custom Performance Results to the Database

```
<task id="13" name="Load Data" tns:taskName="LoadData" >
 <ioSpecification>
     <dataInput id="ToolInput" name="Tool" />
     <dataInput id="ResultsURIInput" name="ResultsURI" />
     <dataInput id="HPCSystemInput" name="HPCSystem" />
     <dataOutput id="ExperimentOutput" name="Experiment" />
     <inputSet>
         <dataInputRefs>ToolInput</dataInputRefs>
         <dataInputRefs>ResultsURIInput</dataInputRefs>
         <dataInputRefs>HPCSystemInput</dataInputRefs>
     </inputSet>
     <outputSet>
         <dataOutputRefs>ExperimentOutput</dataOutputRefs>
     </outputSet>
 </ioSpecification>
 <dataInputAssociation>
     <sourceRef>perfToolVar</sourceRef>
     <targetRef>ToolInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
     <sourceRef>resultsUriVar</sourceRef>
     <targetRef>ResultsURIInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
     <sourceRef>hpcSystemVar</sourceRef>
     <targetRef>HPCSystemInput</targetRef>
 </dataInputAssociation>
 <dataOutputAssociation>
     <sourceRef>ExperimentOutput</sourceRef>
     <targetRef>experimentVar</targetRef>
 </dataOutputAssociation>
</task>
```

## B.10. Store Additional Information to an Experiment

```
<task id="11" name="Add Exp Extra Info" tns:taskName="AddExtraInfo2Experiment" >
 <ioSpecification>
     <dataInput id="ValueInput" name="Value" />
     <dataInput id="ParameterInput" name="Parameter" />
     <dataInput id="ExperimentInput" name="Experiment" />
     <inputSet>
          <dataInputRefs>ValueInput</dataInputRefs>
          <dataInputRefs>ParameterInput</dataInputRefs>
          <dataInputRefs>ExperimentInput</dataInputRefs>
     </inputSet>
     <outputSet> </outputSet>
 </ioSpecification>
 <dataInputAssociation>
     <sourceRef>addInfoParamValue</sourceRef>
     <targetRef>ValueInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
     <sourceRef>addInfoParamName</sourceRef>
     <targetRef>ParameterInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
     <sourceRef>experimentVar</sourceRef>
     <targetRef>ExperimentInput</targetRef>
 </dataInputAssociation>
</task>
```

## B.11. Node for Interactive Questions

```
<task id=”12” name=”Question?” tns:taskName=”AskQuestion” >
 <ioSpecification>
    <dataInput id=”QuestionInput” name=”Question” />
    <dataInput id=”DefaultValueInput” name=”DefaultValue” />
    <dataInput id=”TitleInput” name=”Title” />
    <dataOutput id=”AnswerOutput” name=”Answer” />
    <inputSet>
        <dataInputRefs>QuestionInput</dataInputRefs>
        <dataInputRefs>DefaultValueInput</dataInputRefs>
        <dataInputRefs>TitleInput</dataInputRefs>
    </inputSet>
    <outputSet>
        <dataOutputRefs>AnswerOutput</dataOutputRefs>
    </outputSet>
 </ioSpecification>
 <dataInputAssociation>
    <sourceRef>questionVar</sourceRef>
    <targetRef>QuestionInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>defaultValueVar</sourceRef>
    <targetRef>DefaultValueInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>titleVar</sourceRef>
    <targetRef>TitleInput</targetRef>
 </dataInputAssociation>
 <dataOutputAssociation>
    <sourceRef>AnswerOutput</sourceRef>
    <targetRef>answerVar</targetRef>
 </dataOutputAssociation>
</task>
```

## B.12. Execute a Remote Process

```
<task id="9" name="Remote Process" tns:taskName="StartRemoteProcess" >
 <ioSpecification>
    <dataInput id="CmdArgsInput" name="CmdArgs" />
    <dataInput id="PortInput" name="Port" />
    <dataInput id="SSHKeyInput" name="SSHKey" />
    <dataInput id="UsernameInput" name="Username" />
    <dataInput id="CmdInput" name="Cmd" />
    <dataInput id="HPCSystemInput" name="HPCSystem" />
    <dataInput id="HostnameInput" name="Hostname" />
    <dataOutput id="StdErrorOutput" name="StdError" />
    <dataOutput id="StdOutOutput" name="StdOut" />
    <dataOutput id="ExitValueOutput" name="ExitValue" />
    <inputSet>
        <dataInputRefs>CmdArgsInput</dataInputRefs>
        <dataInputRefs>PortInput</dataInputRefs>
        <dataInputRefs>SSHKeyInput</dataInputRefs>
        <dataInputRefs>UsernameInput</dataInputRefs>
        <dataInputRefs>CmdInput</dataInputRefs>
        <dataInputRefs>HPCSystemInput</dataInputRefs>
        <dataInputRefs>HostnameInput</dataInputRefs>
    </inputSet>
    <outputSet>
        <dataOutputRefs>StdErrorOutput</dataOutputRefs>
        <dataOutputRefs>StdOutOutput</dataOutputRefs>
        <dataOutputRefs>ExitValueOutput</dataOutputRefs>
    </outputSet>
 </ioSpecification>
 <dataInputAssociation>
    <sourceRef>argsToUseVar</sourceRef>
    <targetRef>CmdArgsInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>portVar</sourceRef>
    <targetRef>PortInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>sshKeyVar</sourceRef>
    <targetRef>SSHKeyInput</targetRef>
 </dataInputAssociation>
 <dataInputAssociation>
    <sourceRef>usernameVar</sourceRef>
```

```
      <targetRef>UsernameInput</targetRef>
  </dataInputAssociation>
  <dataInputAssociation>
      <sourceRef>commandToRunVar</sourceRef>
      <targetRef>CmdInput</targetRef>
  </dataInputAssociation>
  <dataInputAssociation>
      <sourceRef>hpcSystemVar</sourceRef>
      <targetRef>HPCSystemInput</targetRef>
  </dataInputAssociation>
  <dataInputAssociation>
      <sourceRef>hostnameVar</sourceRef>
      <targetRef>HostnameInput</targetRef>
  </dataInputAssociation>
  <dataOutputAssociation>
      <sourceRef>StdErrorOutput</sourceRef>
      <targetRef>stdErrorVar</targetRef>
  </dataOutputAssociation>
  <dataOutputAssociation>
      <sourceRef>StdOutOutput</sourceRef>
      <targetRef>stdOutVar</targetRef>
  </dataOutputAssociation>
  <dataOutputAssociation>
      <sourceRef>ExitValueOutput</sourceRef>
      <targetRef>exitValueVar</targetRef>
  </dataOutputAssociation>
</task>
```

Appendix C

# Internal Database Scheme and Data Persistence Classes

## C.1. Scheme of the Internal Database

### C.1.1. Table: Experiment

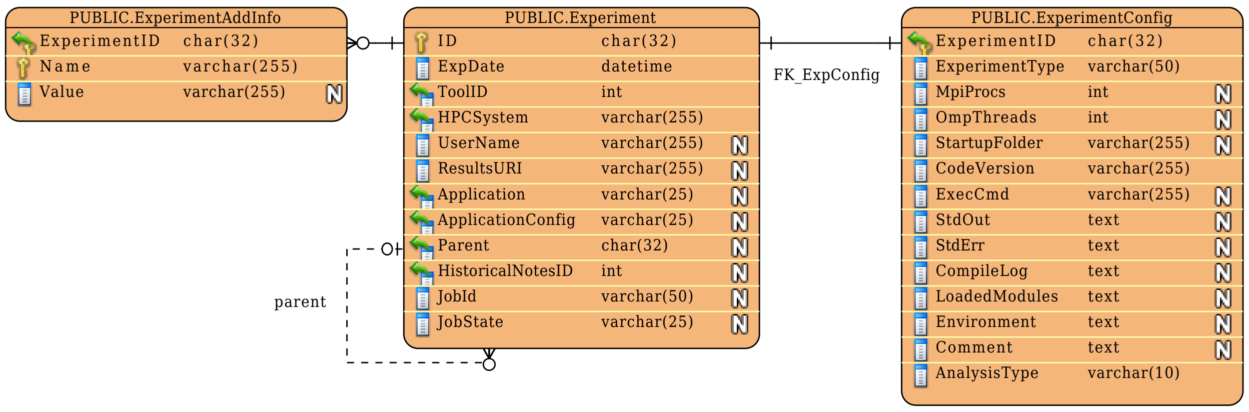


Figure C.1.: Structure of the Experiment table

The main table storing information about PAThWay's experiments is called *Experiment* and contains the following columns:

- ID - this field contains an automatically generated unique id for each new experiment based on the universally unique identifier (UUID) [126] format.
- ExpDate - exact date and time when this experiment was started.

- Application - the first part of a foreign key for interlinking the configuration of the application that was used within the current experiment (see Section 9.2.2).

- ApplicationConfig - the second part of the foreign key for linking to a configuration in the *Applications* table.

- ToolID - another foreign key for identifying the chosen performance engineering tool and its settings which are stored in the *Tools* table (see Section 9.2.4).

- HPCSystem - a foreign key linking the target HPC system that was used throughout the experiment with its full configuration in the *HPCSystems* table (see Section 9.2.3).

- UserName - the user name of the account that triggered the start of this experiment.

- ResultsURI - the location where the results from the current performance engineering experiment are stored. It uses the uniform resource identifier (URI) [23] format in order to uniquely specify the path to the resources.
  Remote resources are specified using `remote://<HPCSystemID>/<path>` whereas local ones using `local://<path>` or directly giving the local path without the scheme part:

```
/home/username/workspace/properties_PerfDynamics_6652.psc
local:///home/username/workspace/properties_PerfDynamics_6652.psc
remote://SuperMIG//home/project1/user3/properties_PerfDynamics_12378.psc
```

- Parent - most of the time, performance engineering studies contain multiple experiments. That is why, this field provides the support for arranging all experiments belonging to a specific study in a hierarchical, parent-child structure in order to group them together. It uses a one-to-many type of relationship between the rows.

- HistoricalNodesID - a foreign key to a table containing user-provided notes about the experiments (see Section 9.2.5).

- JobId - the identification of the underlying parallel job which is used to execute the given experiment as returned by the batch system scheduler system of the chosen machine. Sometimes, this information can be used to obtain further statistics for the execution of an HPC job or cross-matched in resource usage report of user's account.

- JobState - the current state of the underlying parallel job. It can obtain any of the following values: *SUBMITTED*, *RUNNING*, *CANCELED*, or *COMPLETED*.

### C.1.2. Table: ExperimentConfig

ExperimentConfig is the first auxiliary table which stores additional information about the run-time environment of an experiment. The data is structured in the following columns:

- ExperimentID - a one-to-one key connecting the runtime configuration of an experiment with its fundamental data stored in table *Experiment*.
- ExperimentType - a user-provided tag representing the current performance engineering experiment. In case this field is left empty during the configuration of an experiment, its default value is set to *Single-Run Analysis*.
- MpiProcs - number of MPI processes used within the current experiment.
- OmpThreads - number of OpenMP threads that were requested during the experiment.
- StartupFolder - path to the folder where the user's application was initially started.
- CodeVersion - if the chosen application is stored in an Eclipse project and tracking of source code is enabled in PAThWay, then this field shows the version user's code that was used throughout the given experiment. Otherwise, it is set to *UNKNOWN*.
- ExecCmd - exact command and arguments that were used for starting user's application. This attribute is set either during code instrumentation or just before an experiment gets started using the values for the selected user's application from the *ExecLocation* and the *Executable* columns from the Application table and an instrumentation suffix provided by the configuration of the chosen performance engineering tool.
- StdOut - a character large object (CLOB) for storing the standard output stream of the underlying parallel job that was use for the experiment.
- StdErr - another CLOB containing the standard error output generated during the execution of the experiment.
- CompileLog - if (re-)instrumentation of the user's code was required within this experiment, then the output of this process is stored in this text field.
- LoadedModules - a JSON-formatted string specifying all modules that were loaded during the execution of the experiment.
- Environment - a list of all environment variables that were defined at experiment's start up. This field also uses the JSON format for storing the data.

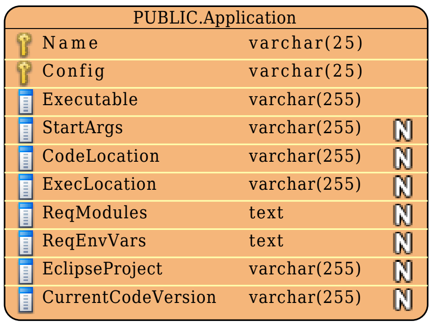
- Comment - a short description about the experiment or a user-provided comment about it. At the start up of any new experiment, PAThWay will automatically ask the user to provide this information as a matter of good practice. However, the framework does not explicitly enforce the provision of a valid comment, thus the user is free to leave it empty.
- AnalysisType - specifies the type of the performance analysis technique that should be used within this experiment. Currently, it can be either *Profile* or *Trace* where the first one is also set as the default value. This field is parsed at runtime by PAThWay to select the correct performance tool configuration.

### C.1.3. Table: ExperimentAddInfo

The second auxiliary table, i.e. *ExperimentAddInfo*, stores user-specified key-value pairs of data which are related to the given experiment. It contains the following three columns:

- ExperimentID - a many-to-one key connecting the user-provided parameters with the fundamental data of an experiment stored in table *Experiment*.
- Name - the string specifying the name of the stored parameter/variable.
- Value - the value of the parameter serialized as a string.

### C.1.4. Table: Application



PUBLIC.Application

| | | | |
|---|---|---|---|
| 🔑 | Name | varchar(25) | |
| 🔑 | Config | varchar(25) | |
| ▤ | Executable | varchar(255) | |
| ▤ | StartArgs | varchar(255) | N |
| ▤ | CodeLocation | varchar(255) | N |
| ▤ | ExecLocation | varchar(255) | N |
| ▤ | ReqModules | text | N |
| ▤ | ReqEnvVars | text | N |
| ▤ | EclipseProject | varchar(255) | N |
| ▤ | CurrentCodeVersion | varchar(255) | N |

Figure C.2.: Structure of the Application table

All the provided by the user configurations of applications are store in the Application table of PAThWay's database. Internally, it is structured as follows:

- Name - a string that identifies a given application. This field is part of the primary key of the table and as such it should create a unique combination when combined with the value from the *Config* column.

- Config - an additional identifier that describes a particular configuration of the given application. For example, this can be the name of an input data set, a custom tag for a combination of runtime setting or the name of the HPC system that contains the application. The only restriction is that this column forms an unique primary key when combined with the *Name* column.

- Executable - this column specifies the name of application's executable without the path to it. The full path will be automatically built using the value from the *ExecLocation* column, this field and an instrumentation suffix provided by the configuration of the chosen performance engineering tool. Additionally, the specified value will be parsed before starting an experiment and any of the placeholders specified in Listing C.1 will be automatically replaced with their corresponding runtime values. Finally, this value will be stored in the *ExecCmd* parameter of the experiment's configuration in order to be used by the framework for starting the application.

  Listing C.1: List of placeholders supported by the runtime configuration of applications

```
{#MPI_PROCS#} – number of MPI processes
{#OMP_THREADS#} – number of OpenMP threads
```

- StartArgs - command-line arguments required for starting the given application. Similarly to the value given in the *Executable* column, the user can parametrize the arguments using the placeholders from Listing C.1.

- CodeLocation - absolute path to the location where the source code of the application is stored. This field is required for the automatic (re-)instrumentation support of the framework.

- ExecLocation - absolute path to the folder which stores the executable of the application after a successful built.

- ReqModules - a JSON-formatted list of modules that should be loaded before the given application is executed.

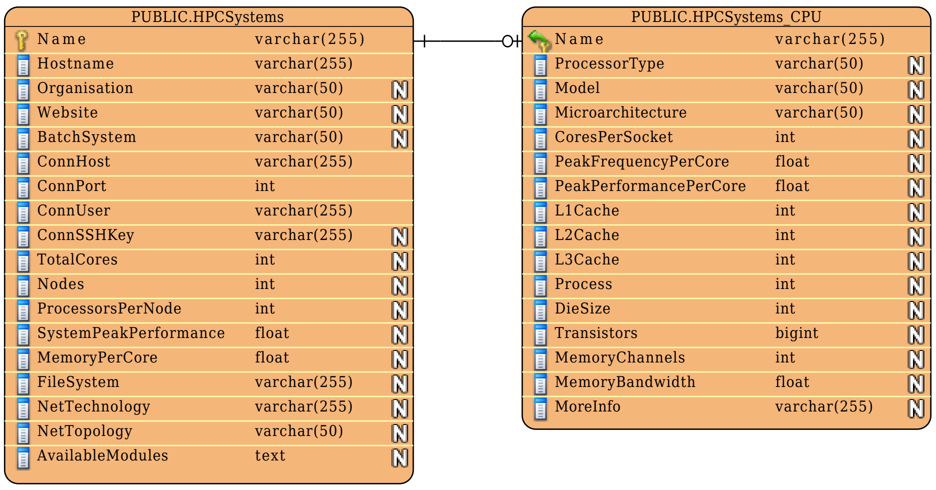
- ReqEnvVars - a list of all environment variables that should be defined before the given application is executed. Similarly to the *Executable* and *StartArgs* parameters, these variables will also be parsed before starting an experiment and any placeholders will be automatically expanded. Internally, it is also stored using the JSON format.

- EclipseProject - if the source code of the application is part of an Eclipse-based project, its name is specified here. This field is then internally used by the code management component of PAThWay in order to provide automatic versioning of the code upon the execution of each new experiment.

- CurrentCodeVersion - if the given application is stored in an Eclipse project and tracking of source code is enabled in PAThWay, then this column contains the current version of the application.

### C.1.5. Table: HPCSystems

Internally, the configuration information about the supported HPC systems is separated in two tables: *HPCSystems* and *HPCSystems_CPU*. The first one stores basic information about the given machine and any login details required to access it. The second one is designed for holding extra information about machine's hardware and software configuration.



| PUBLIC.HPCSystems | | |
|---|---|---|
| Name | varchar(255) | |
| Hostname | varchar(255) | |
| Organisation | varchar(50) | N |
| Website | varchar(50) | N |
| BatchSystem | varchar(50) | N |
| ConnHost | varchar(255) | |
| ConnPort | int | |
| ConnUser | varchar(255) | |
| ConnSSHKey | varchar(255) | N |
| TotalCores | int | N |
| Nodes | int | N |
| ProcessorsPerNode | int | N |
| SystemPeakPerformance | float | N |
| MemoryPerCore | float | N |
| FileSystem | varchar(255) | N |
| NetTechnology | varchar(255) | N |
| NetTopology | varchar(50) | N |
| AvailableModules | text | N |

| PUBLIC.HPCSystems_CPU | | |
|---|---|---|
| Name | varchar(255) | |
| ProcessorType | varchar(50) | N |
| Model | varchar(50) | N |
| Microarchitecture | varchar(50) | N |
| CoresPerSocket | int | N |
| PeakFrequencyPerCore | float | N |
| PeakPerformancePerCore | float | N |
| L1Cache | int | N |
| L2Cache | int | N |
| L3Cache | int | N |
| Process | int | N |
| DieSize | int | N |
| Transistors | bigint | N |
| MemoryChannels | int | N |
| MemoryBandwidth | float | N |
| MoreInfo | varchar(255) | N |

Figure C.3.: Structure of the HPCSystems table

The HPCSystems table has the following structure:

- Hostname - primary internet address of the given HPC machine (might not be directly accessible).

- Organisation - name of the organization administering the given HPC machine.

- Website - a link to a website providing further information about the system.

- BatchSystem - type of underlying scheduling system used by the given machine. This is needed for PAThWay to be able to automatically submit and monitor parallel jobs.
- ConnHost - often the primary address of an HPC system is not directly accessible from the public internet. The users need to use special type of connection forwarding, e.g. SSH tunneling, in order to be able to login. That is why, this field specifies an alternative internet address where such a tunnel is running. The framework will then use this as a primary connection point to communicate with the given machine.
- ConnPort - network port to use for reaching the created SSH tunnel on the previously specified machine.
- ConnUser - username to be used for accessing the remote machine.
- ConnSSHKey - the location of a SSH private that can be used for authentication.
- TotalCores - total number of CPU cores from all nodes available on the machine.
- Nodes - number of separate, distributed-memory nodes.
- ProcessorsPerNode - total number of processors available on each node.
- SystemPeakPerformance [$TFlop/s$] - maximum performance as officially stated by the manufacturer of the machine. This value is most of the time based on the performance of the LINPACK benchmark.
- MemoryPerCore [$GByte$] - total amount of operating memory available to a single core.
- FileSystem - type of the underlying file system (e.g. GPFS, Lustre, NetApp NAS, etc.).
- NetTechnology - type of network communications link used to interconnect the separate nodes (e.g. InfiniBand, Myrinet, Ethernet, etc.).
- NetTopology - underlying network topology in which the nodes are connected using the above techonology (e.g. fat-tree, 3D-Torus, Hypercube, etc.)
- AvailableModules - a list of all currently available software modules on the given machine.

### C.1.6. Table: HPCSystems_CPU

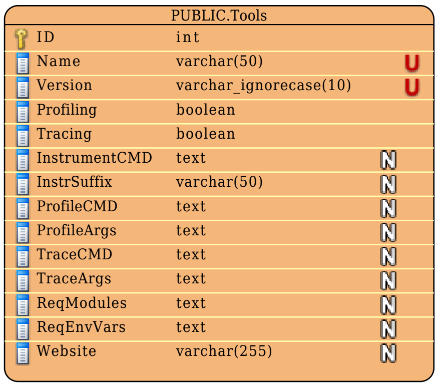
The second table storing data related to HPC systems is the called HPCSystems_CPU. Its scheme contains the following columns:

- Name - a foreign key linking each one of these additional entries to a particular HPC machine from the *HPCSystems* table.
- ProcessorType - exact type of processes used on the given machine (e.g. Intel Xeon Sandy Bridge-EP, Intel Xeon Westmere-EX, etc.).
- Model - exact model from the previously given processor series (e.g. E5-2680, E7-4870, etc.).
- Microarchitecture - code-name of the microarcitecture of the CPU (e.g. Nehalem, Sandy Bridge, etc.).
- CoresPerSocket - total number of CPU cores available on a single die.
- PeakFrequencyPerCore [$MHz$] - maximum CPU frequency
- PeakPerformancePerCore [$GFlop/s$] - maximum theoretical performance per core.
- L1Cache [$KByte$] - size of the L1 processor cache.
- L2Cache [$KByte$] - size of the L2 processor cache.
- L3Cache [$KByte$] - size of the L3 processor cache (if it exists).
- Process - width of the standard data word supported by processor's instruction set (32-bit or 64-bit).
- DieSize [$mm^2$] - total area of the semiconductor die.
- Transistors - total number of transistors available on the die.
- MemoryChannels - total number of memory channels which the memory controller supports.
- MemoryBandwidth [$GB/s$] - maximum memory
- MoreInfo - a website containing further information about the given CPU and its features.

### C.1.7. Table: Tools

The tool-specific configuration data required for automatically instrumenting user's code and starting new performance analysis experiments is stored in the Tools table of the internal database.

The information that should be stored in each of the columns of *Tools* is defined as follows:


PUBLIC.Tools

| | | |
|---|---|---|
| ID | int | |
| Name | varchar(50) | U |
| Version | varchar_ignorecase(10) | U |
| Profiling | boolean | |
| Tracing | boolean | |
| InstrumentCMD | text | N |
| InstrSuffix | varchar(50) | N |
| ProfileCMD | text | N |
| ProfileArgs | text | N |
| TraceCMD | text | N |
| TraceArgs | text | N |
| ReqModules | text | N |
| ReqEnvVars | text | N |
| Website | varchar(255) | N |

Figure C.4.: Structure of the Tools table

- Name - an identifier for the given performance tool. This field serves as the first part of the primary key of the table and as such it should create a unique combination when combined with the value from the *Version* column.

- Version - version og the given tool for which the given settings apply. This field forms the second part of the primary key of the table.

- Profiling - specified whether the performance tool supports profiling-based analysis.

- Tracing - indicates if event tracing is supported by the tool.

- InstrumentCMD - a string specifying the command that has to be used and all its required arguments in order to instrument user's code. This is typically a wrapper script that should be added as a prefix to the compiler invocation during compilation. Any of the placeholders shown in Listing C.2 can be used to parametric the command.

- InstrSuffix - an identification suffix which the framework automatically appends to the name of executable at runtime. This is required in order to differentiate between binaries instrumented with different performance analysis tools. This step also requires modifications to application's build process in order for this suffix to be automatically added at the end of every compilation.

- ProfileCMD - command that has to be used in order to start a new profile-based performance analysis experiment with the selected tool.

- ProfileArgs - list of arguments that must be specified for starting the profiling experiment. Similarly to the *InstrumentCMD*, these arguments can also be parametrized.

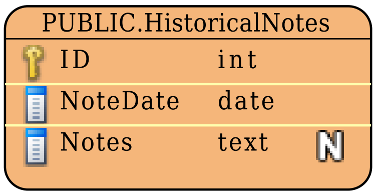
- TraceCMD -a string specifying tool's command for starting a new tracing-based performance analysis.
- TraceArgs - list of required for triggering a new event tracing experiment. Similarly to the *InstrumentCMD* and *ProfileArgs*, these arguments can be parametrized using the placeholders from Listing C.2.
- ReqModules - list of all external module required for the execution of the given tool.
- ReqEnvVars - list of all environment variables required by the given tool.
- Website - a website containing further information about the given performance tool, its features and typical usage scenarios.

Listing C.2: List of variable placeholders supported by the runtime configuration of performance tools

```
{#TOOL_CMD#} – name of the executable of the given performance tool

{#MPI_PROCS#} – number of MPI processes
{#OMP_THREADS#} – number of OpenMP threads
{#MPI_EXEC#} – MPI execution command (e.g. mpirun, mpiexec, poe)

{#APP_NAME#} – path to the executable for the user's application
{#APP_ARGS#} – command–line arguments for the user's application
{#OUTPUT_LOC#} – location of the results from an experiment
```

### C.1.8. Table: HistoricalNotes

All user specified notes for documenting projects and performance engineering experiments are stores in the *HistoricalNotes* table.



| PUBLIC.HistoricalNotes | | |
|---|---|---|
| ID | int | |
| NoteDate | date | |
| Notes | text | N |

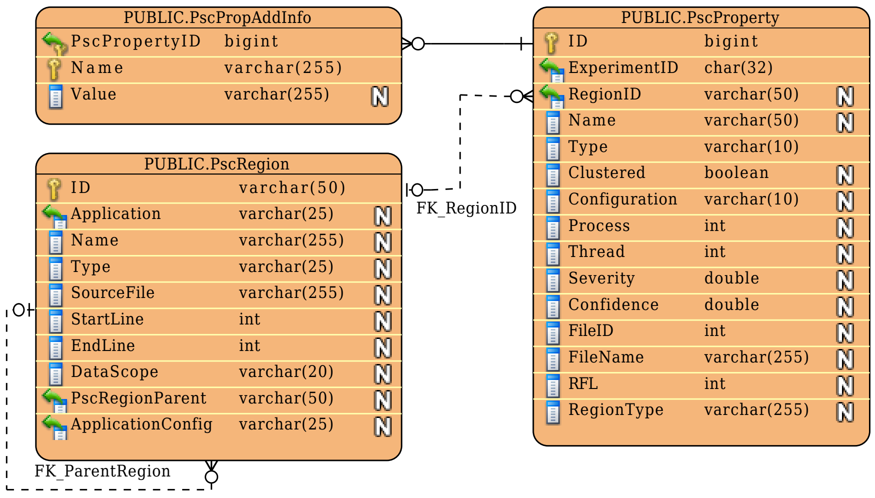
Figure C.5.: Structure of the HistoricalNotes table

This table is defined as follows:

- ID - unique identifier representing the given user-specified note.
- NoteDate - date when a particular note was first created.
- Notes - actual text of the user-specified note stored in a textual form using a Wiki-based formatting language.

## C.1.9. Table: PscProperty

PAThWay provides a set of SQL tables specifically designed to store performance results from the Periscope performance analysis toolkit in order to enhance the performance analysis and data processing step when using that tool.



| PUBLIC.PscPropAddInfo | | |
|---|---|---|
| PscPropertyID | bigint | |
| Name | varchar(255) | |
| Value | varchar(255) | N |

| PUBLIC.PscRegion | | |
|---|---|---|
| ID | varchar(50) | |
| Application | varchar(25) | N |
| Name | varchar(255) | N |
| Type | varchar(25) | N |
| SourceFile | varchar(255) | N |
| StartLine | int | N |
| EndLine | int | N |
| DataScope | varchar(20) | N |
| PscRegionParent | varchar(50) | N |
| ApplicationConfig | varchar(25) | N |

| PUBLIC.PscProperty | | |
|---|---|---|
| ID | bigint | |
| ExperimentID | char(32) | |
| RegionID | varchar(50) | N |
| Name | varchar(50) | N |
| Type | varchar(10) | |
| Clustered | boolean | N |
| Configuration | varchar(10) | N |
| Process | int | N |
| Thread | int | N |
| Severity | double | N |
| Confidence | double | N |
| FileID | int | N |
| FileName | varchar(255) | N |
| RFL | int | N |
| RegionType | varchar(255) | N |

Figure C.6.: Structure of the PscProperty, PscRegion and PscPropAddInfo tables

Internally, the collected performance data is split into three table: *PscProperty*, *PscRegion* and *PscPropAddInfo*. The first one stores all detected bottleneck which the other two contain supportive information. The structure of the PscProperty table is as follows:

- ID - an entity id which is internally generated by the database engine.
- ExperimentID - a foreign key linking this bottleneck of the actual PAThWay's experiment throughout which it was found.
- RegionID - a foreign key representing the relationship of the given property with its corresponding code region from the *PscRegion* table.
- Name - a descriptive name for the given inefficiency.

- Type - an identification tag that uniquely determines the type of the given performance property.
- Clustered - as Periscope supports online clustering of the detected performance bottlenecks, this flag specifies if the given property shows an already aggregated behavior or it represents a single performance bottleneck.
- Configuration - a string specifying the total number of MPI processes and OpenMP threads that were used when this bottleneck was found.
- Process - id of the MPI process where the given performance property was found.
- Thread - OpenMP's thread number that exposed the given performance behavior.
- Severity - impact of the given bottleneck on the overall execution of the application.
- Confidence - level of certainty of the result.
- FileID - an internal id of the source file that contains the given performance bottleneck. It is used by Periscope as mapping key between code regions and detected properties.
- FileName - name of the source file which contains the given property.
- RFL - first line of the code region which contains the given property.
- RegionType - a tag specifying the type of the code region where the bottleneck was found (e.g. *USER_REGION*, *LOOP_REGION*, etc.).

### C.1.10. Table: PscRegion

The first auxiliary table for storing performance results from the Periscope toolkit is called PscRegion. It is designed to hold the data from the Standardized Intermediate Representation (SIR) files used by Periscope. Its database schema contains the following columns:

- ID - an identification tag that uniquely determines the given code region.
- Application - the first part of a foreign key linking this code region to the internal configuration entry of used application stored in table *Application*.
- ApplicationConfig - the second part of the foreign key linking to the *Application* table.
- Name - a string giving additional information about the code region. In case it is a region representing a function, its name is specified in this column.

- Type - the type of the current code region (e.g. loop, parallelLoop, subroutine, etc.).
- SourceFile - name of the source file which contains the given code region.
- StartLine - first line of the code region.
- EndLine - last line of the code region.
- DataScope - this column specifies additional data-related flags.
- PscRegionParent - refer to the parent of this code region in order to represent the typically hierarchical structure of SIR files.

### C.1.11. Table: PscPropAddInfo

The second auxiliary table for Periscope data is the PscPropAddInfo. It stores any provided by the performance properties data such as single values or complete time series. Its is generically designed in order to accommodate all possible data types which this extra information can have. Its three columns are defined as follows:

- PscPropertyID - a many-to-one foreign key linking a set of entries in this table to their original performance properties stored in *PscProperty*.
- Name - a string which uniquely identifies the current entry among all other additional data that were provided with a single property.
- Value - actual value of the entry in a serialized format.

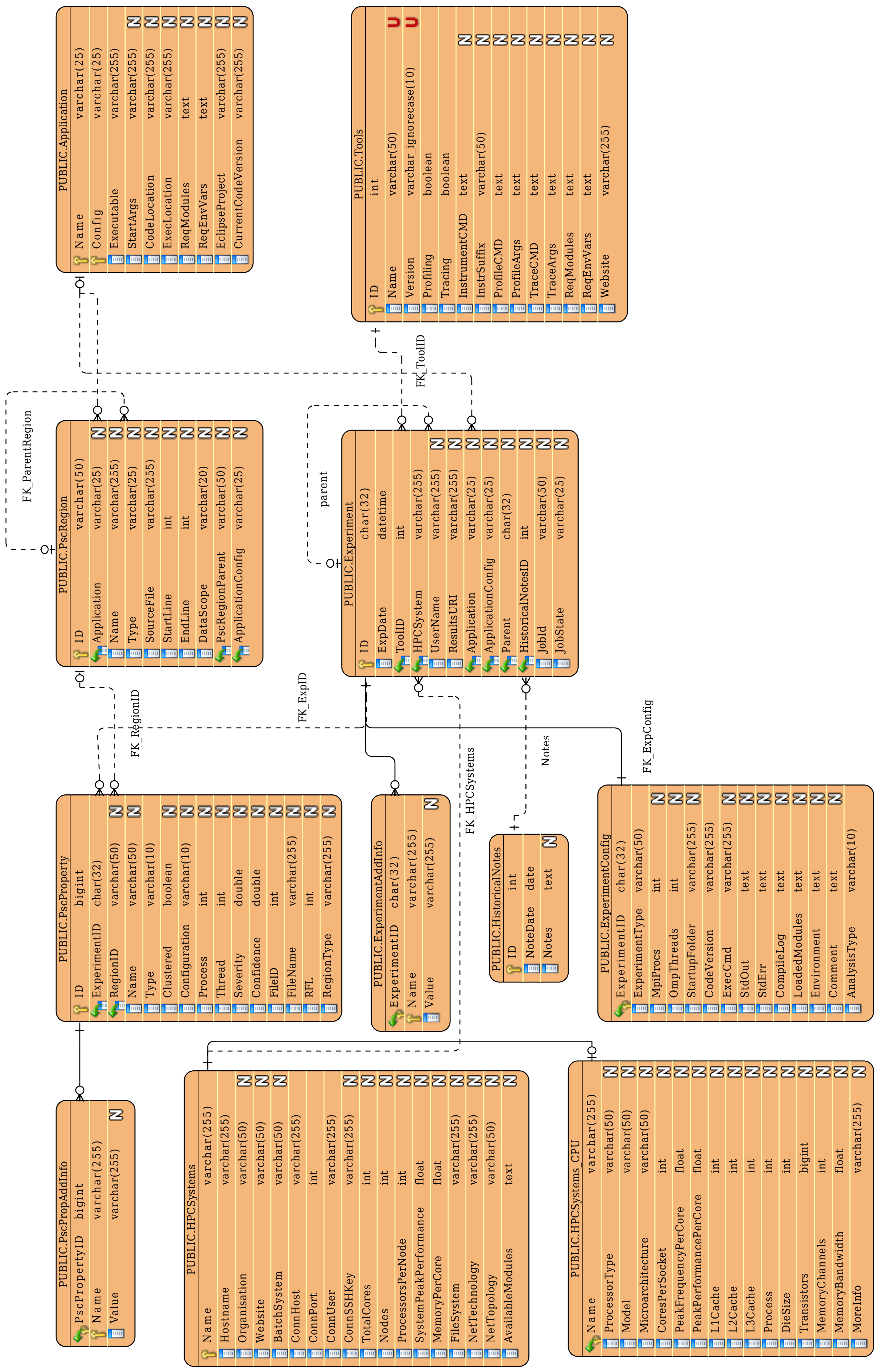


Figure C.7.: Scheme of PAThWay's internal database

## C.2. Data Persistence Classes

Listing C.3: An example Hibernate's configuration of the mapping between the Tools objects and the corresponding database table

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="pathway.data.persistence.Tools" table="Tools" schema="PUBLIC" lazy="false">
        <id name="ID" column="ID" type="integer" unsaved-value="0">
                <generator class="increment">
                        <param name="schema">PUBLIC</param>
                </generator>
        </id>

        <property name="name" column="Name" type="string" length="50" not-null="true" lazy="false"/>
        <property name="version" column="Version" type="string" length="10" not-null="true" lazy="false"/>
        <property name="profiling" type="boolean" not-null="true" lazy="false">
                <column name="Profiling" default="false"/>
        </property>
        <property name="tracing" type="boolean" not-null="true" lazy="false">
                <column name="Tracing" default="false"/>
        </property>
        <property name="instrumentCMD" column="InstrumentCMD" type="text" not-null="false" lazy="false"/>
        <property name="instrSuffix" column="InstrSuffix" type="string" length="50" not-null="false" lazy="false"/>
        <property name="profileCMD" column="ProfileCMD" type="text" not-null="false" lazy="false"/>
        <property name="profileArgs" column="ProfileArgs" type="text" not-null="false" lazy="false"/>
        <property name="traceCMD" column="TraceCMD" type="text" not-null="false" lazy="false"/>
        <property name="traceArgs" column="TraceArgs" type="text" not-null="false" lazy="false"/>
        <property name="reqModules" column="ReqModules" type="text" not-null="false" lazy="false"/>
        <property name="reqEnvVars" column="ReqEnvVars" type="text" not-null="false" lazy="false"/>
        <property name="website" column="Website" type="string" length="255" not-null="false" lazy="false"/>

        <set name="experiment" lazy="extra" cascade="save-update,lock" inverse="true">
                <key column="ToolID" not-null="true"/>
                <one-to-many class="pathway.data.persistence.Experiment"/>
        </set>
  </class>
</hibernate-mapping>
```

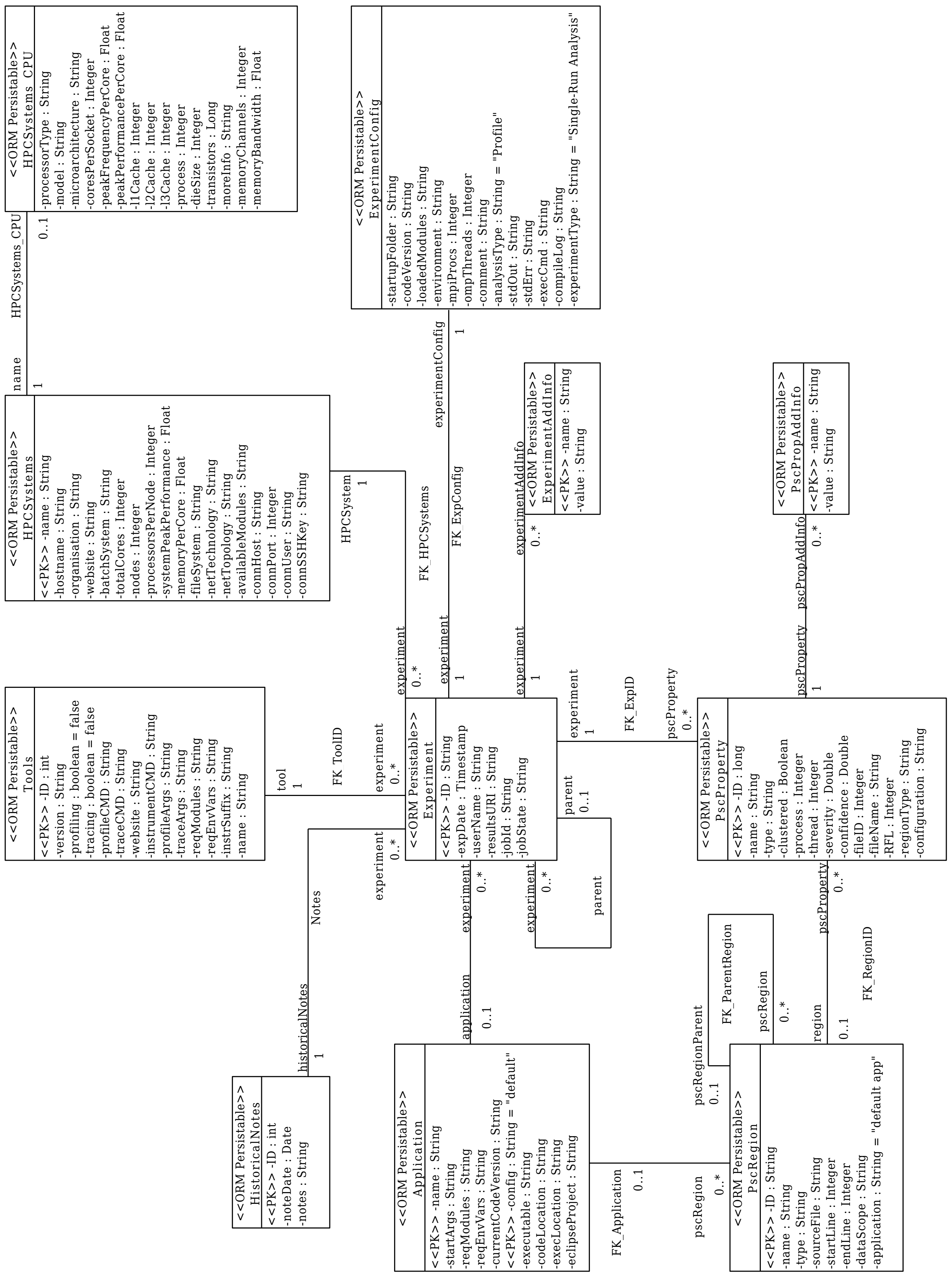


Figure C.8.: Data Persistence Classes (ORM)

Appendix D

# Documentation Module: Modifications to EclipseWiki

## D.1. PAThWay Extension Classes

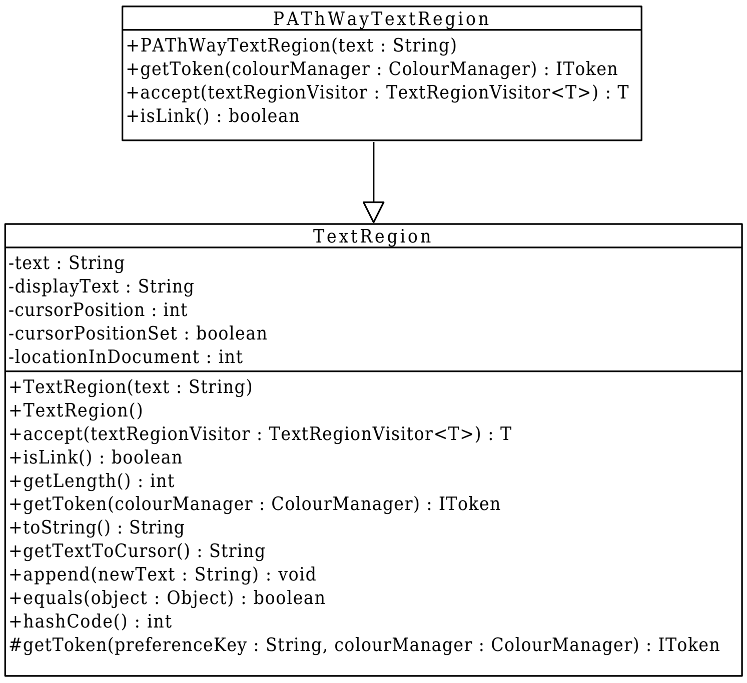

PAThWayTextRegion
+PAThWayTextRegion(text : String)
+getToken(colourManager : ColourManager) : IToken
+accept(textRegionVisitor : TextRegionVisitor<T>) : T
+isLink() : boolean
TextRegion
-text : String
-displayText : String
-cursorPosition : int
-cursorPositionSet : boolean
-locationInDocument : int
+TextRegion(text : String)
+TextRegion()
+accept(textRegionVisitor : TextRegionVisitor<T>) : T
+isLink() : boolean
+getLength() : int
+getToken(colourManager : ColourManager) : IToken
+toString() : String
+getTextToCursor() : String
+append(newText : String) : void
+equals(object : Object) : boolean
+hashCode() : int
#getToken(preferenceKey : String, colourManager : ColourManager) : IToken

Figure D.1.: Structure of the PAThWayTextRegion class

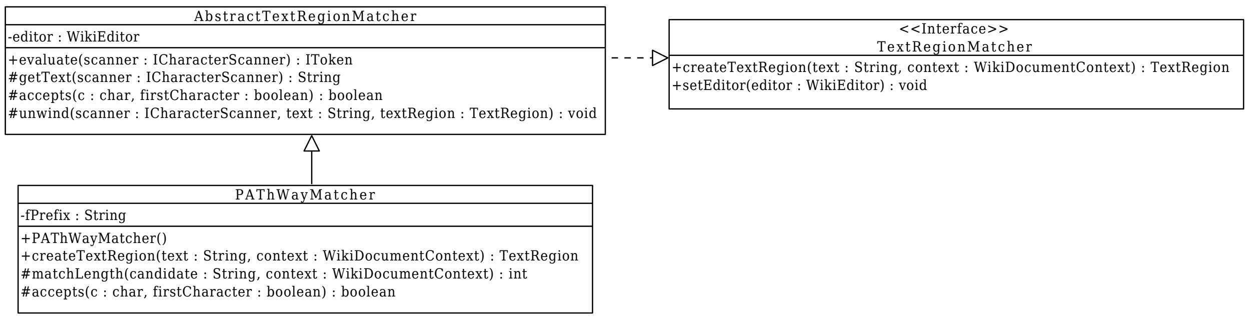

AbstractTextRegionMatcher
-editor : WikiEditor
+evaluate(scanner : ICharacterScanner) : IToken
#getText(scanner : ICharacterScanner) : String
#accepts(c : char, firstCharacter : boolean) : boolean
#unwind(scanner : ICharacterScanner, text : String, textRegion : TextRegion) : void
<<Interface>>
TextRegionMatcher
+createTextRegion(text : String, context : WikiDocumentContext) : TextRegion
+setEditor(editor : WikiEditor) : void
PAThWayMatcher
-fPrefix : String
+PAThWayMatcher()
+createTextRegion(text : String, context : WikiDocumentContext) : TextRegion
#matchLength(candidate : String, context : WikiDocumentContext) : int
#accepts(c : char, firstCharacter : boolean) : boolean

Figure D.2.: Structure of the PAThWayTextRegionMatcher class

## D.2. Modifications to EclipseWiki's Internal Classes

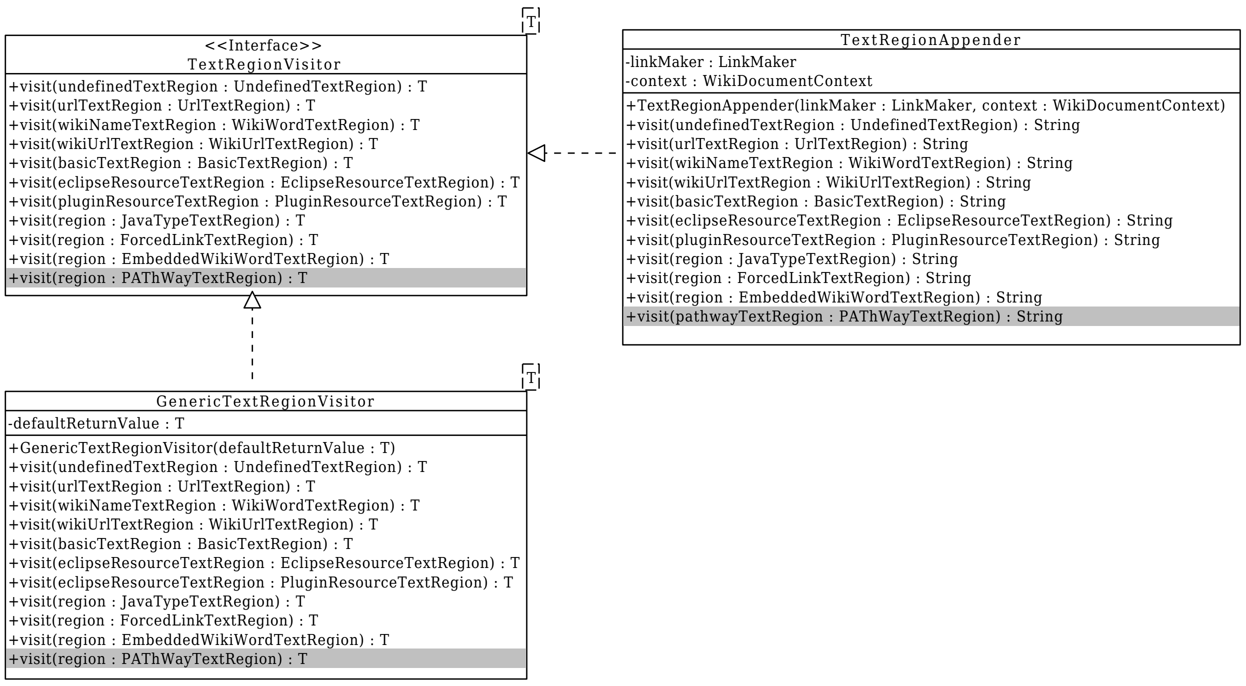
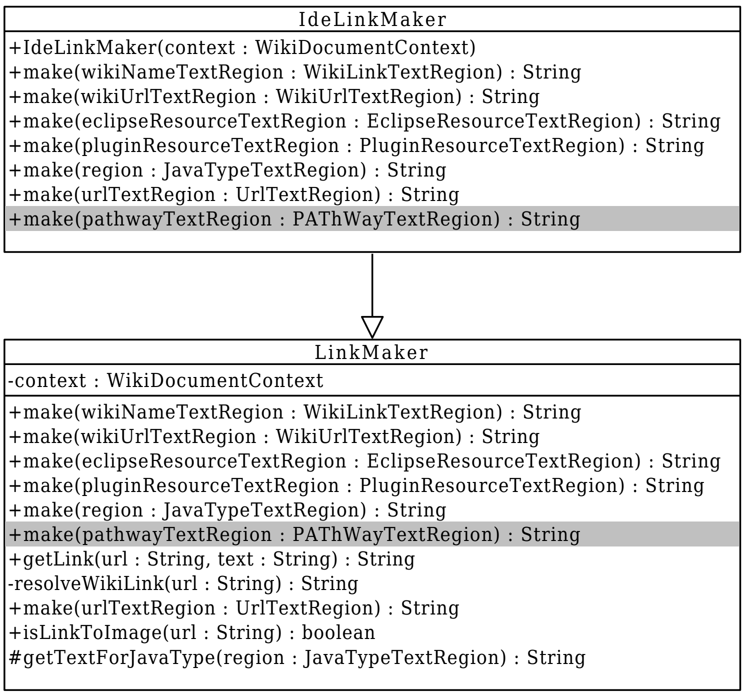
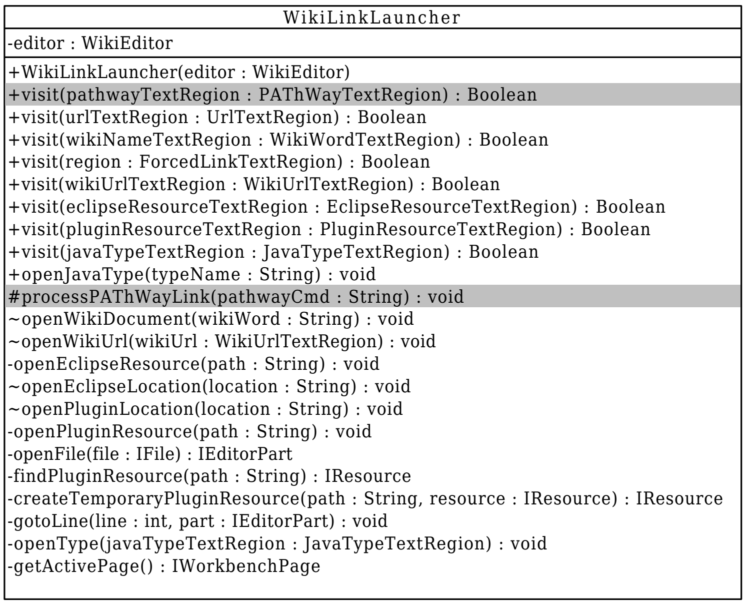

T
<<Interface>>
TextRegionVisitor
+visit(undefinedTextRegion : UndefinedTextRegion) : T
+visit(urlTextRegion : UrlTextRegion) : T
+visit(wikiNameTextRegion : WikiWordTextRegion) : T
+visit(wikiUrlTextRegion : WikiUrlTextRegion) : T
+visit(basicTextRegion : BasicTextRegion) : T
+visit(eclipseResourceTextRegion : EclipseResourceTextRegion) : T
+visit(pluginResourceTextRegion : PluginResourceTextRegion) : T
+visit(region : JavaTypeTextRegion) : T
+visit(region : ForcedLinkTextRegion) : T
+visit(region : EmbeddedWikiWordTextRegion) : T
+visit(region : PAThWayTextRegion) : T
TextRegionAppender
-linkMaker : LinkMaker
-context : WikiDocumentContext
+TextRegionAppender(linkMaker : LinkMaker, context : WikiDocumentContext)
+visit(undefinedTextRegion : UndefinedTextRegion) : String
+visit(urlTextRegion : UrlTextRegion) : String
+visit(wikiNameTextRegion : WikiWordTextRegion) : String
+visit(wikiUrlTextRegion : WikiUrlTextRegion) : String
+visit(basicTextRegion : BasicTextRegion) : String
+visit(eclipseResourceTextRegion : EclipseResourceTextRegion) : String
+visit(pluginResourceTextRegion : PluginResourceTextRegion) : String
+visit(region : JavaTypeTextRegion) : String
+visit(region : ForcedLinkTextRegion) : String
+visit(region : EmbeddedWikiWordTextRegion) : String
+visit(pathwayTextRegion : PAThWayTextRegion) : String
T
GenericTextRegionVisitor
-defaultReturnValue : T
+GenericTextRegionVisitor(defaultReturnValue : T)
+visit(undefinedTextRegion : UndefinedTextRegion) : T
+visit(urlTextRegion : UrlTextRegion) : T
+visit(wikiNameTextRegion : WikiWordTextRegion) : T
+visit(wikiUrlTextRegion : WikiUrlTextRegion) : T
+visit(basicTextRegion : BasicTextRegion) : T
+visit(eclipseResourceTextRegion : EclipseResourceTextRegion) : T
+visit(eclipseResourceTextRegion : PluginResourceTextRegion) : T
+visit(region : JavaTypeTextRegion) : T
+visit(region : ForcedLinkTextRegion) : T
+visit(region : EmbeddedWikiWordTextRegion) : T
+visit(region : PAThWayTextRegion) : T

Figure D.3.: Structure of the TextRegionVisitor classes



```
IdeLinkMaker
+IdeLinkMaker(context : WikiDocumentContext)
+make(wikiNameTextRegion : WikiLinkTextRegion) : String
+make(wikiUrlTextRegion : WikiUrlTextRegion) : String
+make(eclipseResourceTextRegion : EclipseResourceTextRegion) : String
+make(pluginResourceTextRegion : PluginResourceTextRegion) : String
+make(region : JavaTypeTextRegion) : String
+make(urlTextRegion : UrlTextRegion) : String
+make(pathwayTextRegion : PAThWayTextRegion) : String

LinkMaker
-context : WikiDocumentContext
+make(wikiNameTextRegion : WikiLinkTextRegion) : String
+make(wikiUrlTextRegion : WikiUrlTextRegion) : String
+make(eclipseResourceTextRegion : EclipseResourceTextRegion) : String
+make(pluginResourceTextRegion : PluginResourceTextRegion) : String
+make(region : JavaTypeTextRegion) : String
+make(pathwayTextRegion : PAThWayTextRegion) : String
+getLink(url : String, text : String) : String
-resolveWikiLink(url : String) : String
+make(urlTextRegion : UrlTextRegion) : String
+isLinkToImage(url : String) : boolean
#getTextForJavaType(region : JavaTypeTextRegion) : String
```

Figure D.4.: Structure of the LinkMaker classes



```
WikiLinkLauncher
-editor : WikiEditor
+WikiLinkLauncher(editor : WikiEditor)
+visit(pathwayTextRegion : PAThWayTextRegion) : Boolean
+visit(urlTextRegion : UrlTextRegion) : Boolean
+visit(wikiNameTextRegion : WikiWordTextRegion) : Boolean
+visit(region : ForcedLinkTextRegion) : Boolean
+visit(wikiUrlTextRegion : WikiUrlTextRegion) : Boolean
+visit(eclipseResourceTextRegion : EclipseResourceTextRegion) : Boolean
+visit(pluginResourceTextRegion : PluginResourceTextRegion) : Boolean
+visit(javaTypeTextRegion : JavaTypeTextRegion) : Boolean
+openJavaType(typeName : String) : void
#processPAThWayLink(pathwayCmd : String) : void
~openWikiDocument(wikiWord : String) : void
~openWikiUrl(wikiUrl : WikiUrlTextRegion) : void
-openEclipseResource(path : String) : void
~openEclipseLocation(location : String) : void
~openPluginLocation(location : String) : void
-openPluginResource(path : String) : void
-openFile(file : IFile) : IEditorPart
-findPluginResource(path : String) : IResource
-createTemporaryPluginResource(path : String, resource : IResource) : IResource
-gotoLine(line : int, part : IEditorPart) : void
-openType(javaTypeTextRegion : JavaTypeTextRegion) : void
-getActivePage() : IWorkbenchPage
```

Figure D.5.: Structure of the WikiLinkLauncher class

## D.3. Example of a Wiki Markup of a Historical Note

```
'''Experiments for 31. December 2012'''
*Scalability Analysis - PAThWay:Exp:ff8081813b675ba8013b675be8a70000
 Application: NPB-BT-MZ - ClassC
 Tool: Periscope
 Number of sub-experiments: 9
**MPI: 4 / OpenMP: 1 ( SuperMUC Fat Node ): ff8081813ef1195d013ef11eee060004
**MPI: 4 / OpenMP: 2 ( SuperMUC Fat Node ): ff8081813ef1195d013ef11f62a80005
**MPI: 8 / OpenMP: 1 ( SuperMUC Fat Node ): ff8081813ef1195d013ef1205f730006
**MPI: 8 / OpenMP: 2 ( SuperMUC Fat Node ): ff8081813ef1195d013ef12088880007
**MPI: 16 / OpenMP: 1 ( SuperMUC Fat Node ): ff8081813ef1195d013ef12159250008
**MPI: 16 / OpenMP: 2 ( SuperMUC Fat Node ): ff8081813ef1195d013ef12183bc0009
**MPI: 32 / OpenMP: 1 ( SuperMUC Fat Node ): ff8081813ef1195d013ef122534f000a
**MPI: 32 / OpenMP: 2 ( SuperMUC Fat Node ): ff8081813ef1195d013ef1227e55000b
**MPI: 32 / OpenMP: 4 ( SuperMUC Fat Node ): ff8081813b675ba8013b623867f10003

----
----

'''PAThWay Sample Experiment Note'''

After completing the scalability analysis experiment, the following graph was generated:
Eclipse:EclipseWikiTest/Wiki/NPB_Measurements_Main-Scalability-Line.jpeg

The exact results can be found in the following table:

|'''''Region'''''|'''''16x1'''''|'''''32x1'''''|'''''32x4'''''|
|Main |98.10| 52.45| 13.64|
|ADI |94.15| 47.53| 12.96|
|Exch_QBC|4.78 | 5.63 | 0.51 |
|MPI |4.39 | 6.73 | 0.48 |
```

Appendix E

# Other Supportive Modules

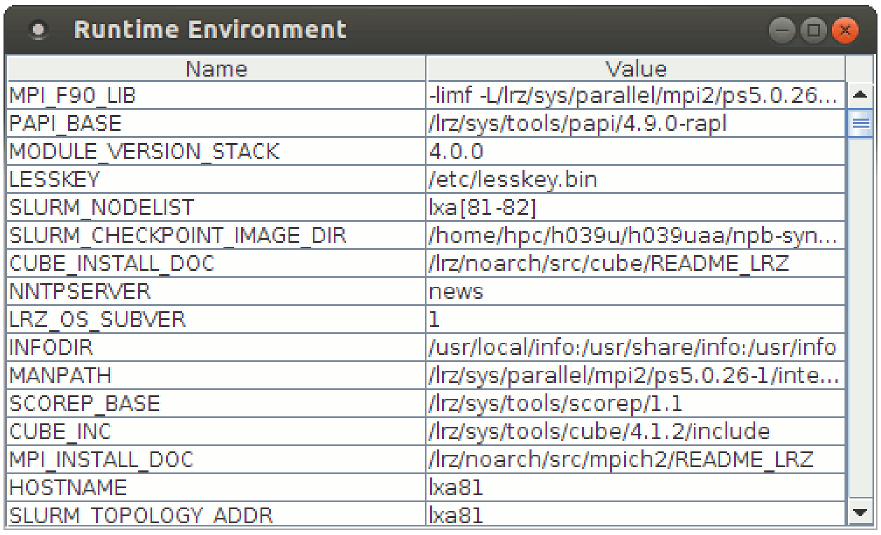
## E.1. Parallel Tools Platform Interface Module

Listing E.1: An excerpt from a PAThWay's resource manager configuration showing the modifications to the standard script for submitting jobs to a SLURM scheduler

```
<script insertEnvironmentAfter="0" deleteAfterSubmit="true">
    <file-staging-location>${ptp_rm:directory#value}</file-staging-location>
    <line><arg>#!/bin/bash</arg></line>
    <line>
        <arg isUndefinedIfMatches="#SBATCH --account=">#SBATCH --account=${ptp_rm:account#value}</arg>
    </line>
    <line>
        <arg isUndefinedIfMatches="#SBATCH --clusters=">#SBATCH --clusters=${ptp_rm:clusters#value}</arg>
    </line>
    <line>
        <arg isUndefinedIfMatches="#SBATCH --ntasks=">#SBATCH --ntasks=${ptp_rm:srun.ntasks#value}</arg>
    </line>
    <line>
        <arg isUndefinedIfMatches="#SBATCH --cpus-per-task=">
            #SBATCH --cpus-per-task=${ptp_rm:ompNumberOfThreads#value}
        </arg>
    </line>
    <line>
        <arg isUndefinedIfMatches="#SBATCH --nodes=">#SBATCH --nodes=${ptp_rm:nodes#value}</arg>
    </line>
    <line>
        <arg isUndefinedIfMatches="#SBATCH --workdir=">#SBATCH --workdir=${ptp_rm:workdir#value}</arg>
    </line>
    <line>
        <arg isUndefinedIfMatches="#SBATCH --output=">#SBATCH --output=${ptp_rm:output#value}</arg>
    </line>
    <line>
```

```
        <arg isUndefinedIfMatches="#SBATCH --error=">#SBATCH --error=${ptp_rm:error#value}</arg>
    </line>
    <line>
        <arg isUndefinedIfMatches="#SBATCH --export=">#SBATCH --export=${ptp_rm:export#value}</arg>
    </line>
    <line>
        <arg isUndefinedIfMatches="#SBATCH --get-user-env=">
            #SBATCH --get-user-env=${ptp_rm:get_user_env#value}
        </arg>
    </line>
    <line>
        <arg isUndefinedIfMatches='#SBATCH --job-name=""'>
            #SBATCH --job-name="${ptp_rm:job_name#value}"
        </arg>
    </line>
    <line>
        <arg isUndefinedIfMatches="#SBATCH --time=">#SBATCH --time=${ptp_rm:time#value}</arg>
    </line>
    <line>
        <arg isUndefinedIfMatches="#SBATCH --mail-type=">
            #SBATCH --mail-type=${ptp_rm:mail_type#value}
        </arg>
    </line>
    <line>
        <arg isUndefinedIfMatches="#SBATCH --mail-user=">
            #SBATCH --mail-user=${ptp_rm:mail_user#value}
        </arg>
    </line>
    <line><arg isUndefinedIfMatches="#SBATCH">#SBATCH ${ptp_rm:verbose#value}</arg></line>

    <!-- PAThWay Additions -->
    <line><arg>source /etc/profile.d/modules.sh</arg></line>
    <line><arg isUndefinedIfMatches="cd">cd ${ptp_rm:workdir#value}</arg></line>
    <line>
        <arg isUndefinedIfMatches="module load ">module load ${ptp_rm:pathway.modules#value}</arg>
    </line>
    <line>
        <arg resolve="true">${ptp_rm:directory#value}/getEnv.sh</arg>
        <arg resolve="false">--output-file=pathway.${SLURM_JOB_ID}.env</arg>
        <arg>--cmd='${ptp_rm:executablePath#value}'</arg>
        <arg>--args='${ptp_rm:progArgs#value}'</arg>
    </line>
    <line>
        <arg attribute="pathway.toolrun" isUndefinedIfMatches="true">${ptp_rm:mpiCommand#value}</arg>
        <arg attribute="pathway.toolrun" isUndefinedIfMatches="true">-n ${ptp_rm:srun.ntasks#value}</arg>
        <arg resolve="true">${ptp_rm:executablePath#value} ${ptp_rm:progArgs#value}</arg>
    </line>
    <!-- PAThWay Additions END -->
</script>
```

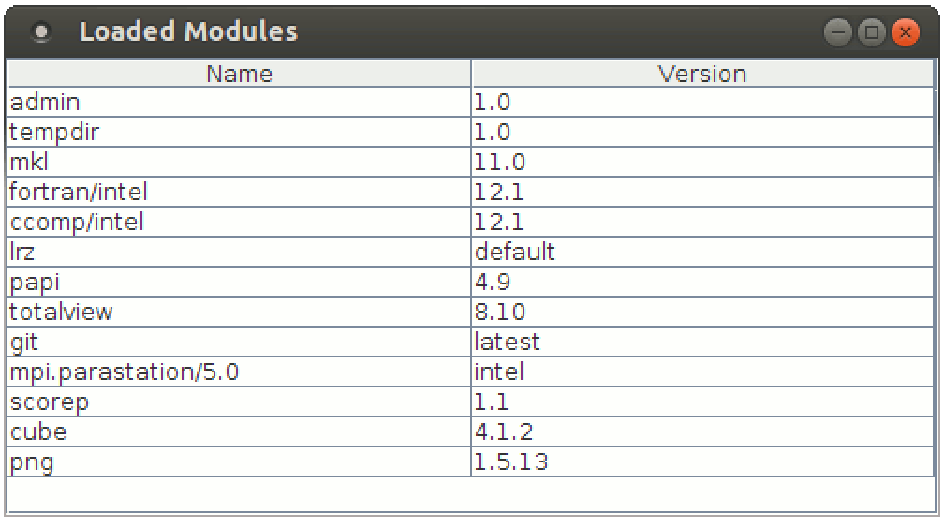
## E.2. Experiments Browser GUI



Runtime Environment

| Name | Value |
|---|---|
| MPI_F90_LIB | -limf -L/lrz/sys/parallel/mpi2/ps5.0.26... |
| PAPI_BASE | /lrz/sys/tools/papi/4.9.0-rapl |
| MODULE_VERSION_STACK | 4.0.0 |
| LESSKEY | /etc/lesskey.bin |
| SLURM_NODELIST | lxa[81-82] |
| SLURM_CHECKPOINT_IMAGE_DIR | /home/hpc/h039u/h039uaa/npb-syn... |
| CUBE_INSTALL_DOC | /lrz/noarch/src/cube/README_LRZ |
| NNTPSERVER | news |
| LRZ_OS_SUBVER | 1 |
| INFODIR | /usr/local/info:/usr/share/info:/usr/info |
| MANPATH | /lrz/sys/parallel/mpi2/ps5.0.26-1/inte... |
| SCOREP_BASE | /lrz/sys/tools/scorep/1.1 |
| CUBE_INC | /lrz/sys/tools/cube/4.1.2/include |
| MPI_INSTALL_DOC | /lrz/noarch/src/mpich2/README_LRZ |
| HOSTNAME | lxa81 |
| SLURM_TOPOLOGY_ADDR | lxa81 |

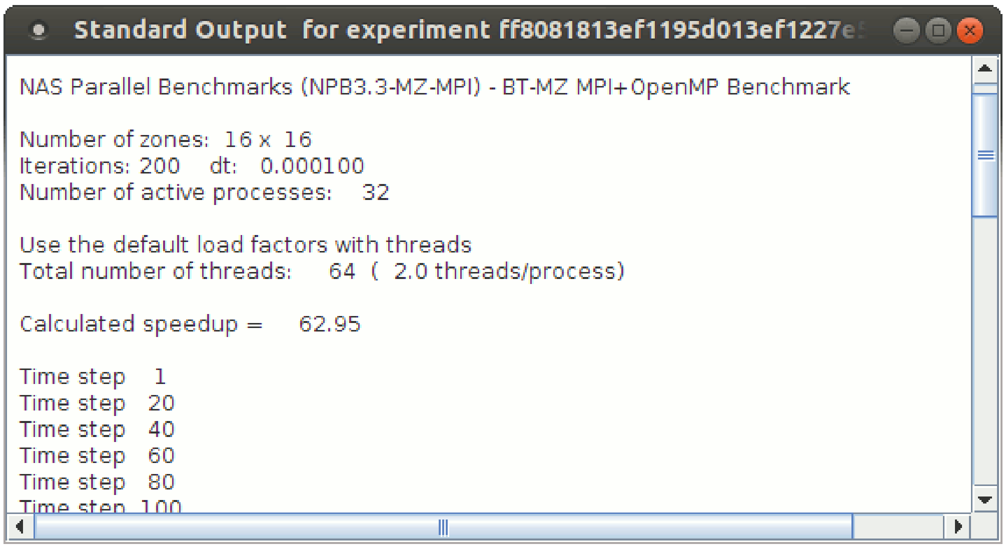
Figure E.1.: User interface for showing all runtime environment variables



Loaded Modules

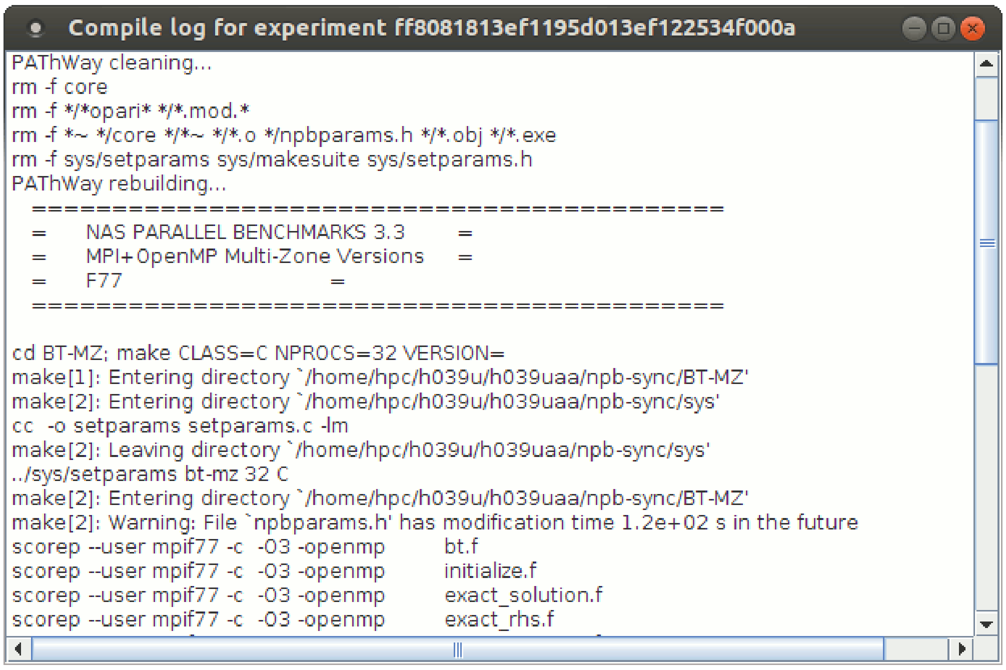
| Name | Version |
|---|---|
| admin | 1.0 |
| tempdir | 1.0 |
| mkl | 11.0 |
| fortran/intel | 12.1 |
| ccomp/intel | 12.1 |
| lrz | default |
| papi | 4.9 |
| totalview | 8.10 |
| git | latest |
| mpi.parastation/5.0 | intel |
| scorep | 1.1 |
| cube | 4.1.2 |
| png | 1.5.13 |

Figure E.2.: User interface for showing the loaded external modules



```
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

Number of zones:  16 x  16
Iterations: 200    dt:   0.000100
Number of active processes:     32

Use the default load factors with threads
Total number of threads:     64  (  2.0 threads/process)

Calculated speedup =     62.95

Time step    1
Time step   20
Time step   40
Time step   60
Time step   80
```

Figure E.3.: User interface for showing the standard output log



```
PAThWay cleaning...
rm -f core
rm -f */*opari* */*.mod.*
rm -f *~ */core */*~ */*.o */npbparams.h */*.obj */*.exe
rm -f sys/setparams sys/makesuite sys/setparams.h
PAThWay rebuilding...
  ============================================
  =      NAS PARALLEL BENCHMARKS 3.3         =
  =      MPI+OpenMP Multi-Zone Versions      =
  =      F77                                 =
  ============================================

cd BT-MZ; make CLASS=C NPROCS=32 VERSION=
make[1]: Entering directory `/home/hpc/h039u/h039uaa/npb-sync/BT-MZ'
make[2]: Entering directory `/home/hpc/h039u/h039uaa/npb-sync/sys'
cc  -o setparams setparams.c -lm
make[2]: Leaving directory `/home/hpc/h039u/h039uaa/npb-sync/sys'
../sys/setparams bt-mz 32 C
make[2]: Entering directory `/home/hpc/h039u/h039uaa/npb-sync/BT-MZ'
make[2]: Warning: File `npbparams.h' has modification time 1.2e+02 s in the future
scorep --user mpif77 -c  -O3 -openmp        bt.f
scorep --user mpif77 -c  -O3 -openmp        initialize.f
scorep --user mpif77 -c  -O3 -openmp        exact_solution.f
scorep --user mpif77 -c  -O3 -openmp        exact_rhs.f
```

Figure E.4.: User interface for showing the compilation log after (re-) instrumentation

Appendix F

# Bibliography

[1] A. Abran and P. Bourque, SWEBOK: Guide to the software engineering Body of Knowledge. IEEE Computer Society, 2004.

[2] Adaptive Computing, Inc, "The MOAB workload manager and the TORQUE resource manager." [Online]. Available: http://www.adaptivecomputing.com/products/open-source/torque/

[3] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "HPCToolkit: Tools for performance analysis of optimized parallel programs," Concurrency and Computation: Practice and Experience, vol. 22, no. 6, pp. 685–701, 2009.

[4] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau et al., "Web services human task (WS-HumanTask)," 2007.

[5] A. Aitken and V. Ilango, "A comparative analysis of traditional software engineering and agile software development," in 46th Hawaii International Conference on System Sciences (HICSS), 2013, pp. 4751–4760.

[6] J. Alameda and J. L. Overbey, "The Eclipse Parallel Tools Platform (PTP): Towards an integrated development environment for improved software engineering on Crays," CUG 2012 Conference Proceedings, 2012.

[7] T. Allweyer, BPMN 2.0: Introduction to the standard for business process modeling. Books on Demand, 2010.

[8] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows," in Proceedings of the 16th International Conference on Scientific and Statistical Database Management. IEEE, 2004, pp. 423–424.

[9] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte et al., "Business process execution language for web services (BPEL4WS)," Microsoft, IBM, Siebel Systems, BEA, and SAP, version, vol. 1, 2003.

[10] C. Aniszczyk, C. Halstrick, C. Ranger, D. Borowitz, G. Wagenknecht, K. Sawicki, M. Kinzler, M. Sohn, R. Rosenberg, R. Stocker, S. Pearce, S. Lay, and S. Zivkov, "JGit - lightweight Java implementation of Git," 2013. [Online]. Available: http://www.eclipse.org/jgit/

[11] Apache Software Foundation, "Apache Derby," 2013. [Online]. Available: http://db.apache.org/derby/

[12] Appian, "Appian BPM Suite: Tapping the power of intelligent processes in the mobile, cloud and social age," Appian Corporation, Executive Summary, March 2012. [Online]. Available: http://www.appian.com/bpm-software/bpm-suite.jsp

[13] Appian, "Appian BPM Suite (version 6.6)," 2013. [Online]. Available: http://www.appian.com/bpm-software/bpm-suite.jsp

[14] R. Ausbrooks, S. Buswell, D. Carlisle, G. Chavchanidze, S. Dalmas, S. Devitt, A. Diaz, S. Dooley, R. Hunter, P. Ion et al., "Mathematical Markup Language (MathML) version 2.0 . W3C recommendation," World Wide Web Consortium, vol. 2003, 2003.

[15] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The NAS parallel benchmarks summary and preliminary results," in Proceedings of the 1991 ACM/IEEE Conference on Supercomputing., 1991, pp. 158 –165.

[16] A. Barker and J. Hemert, "Scientific workflow: A survey and research directions," in Parallel Processing and Applied Mathematics, ser. Lecture Notes in Computer Science, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, Eds. Springer Berlin Heidelberg, 2008, vol. 4967, pp. 746–753.

[17] D. Barry and T. Stanienda, "Solving the Java object storage problem," Computer, vol. 31, no. 11, pp. 33–40, 1998.

[18] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC storage resource broker," in Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative Research, ser. CASCON '98. IBM Press, 1998, pp. 5–.

[19] K. Beck, Test driven development: By example. Addison-Wesley Professional, 2003.

[20] K. Beck and C. Andres, Extreme programming explained: embrace change. Addison-Wesley Professional, 2004.

[21] R. Bell, A. Malony, and S. Shende, "Paraprof: A portable, extensible, and scalable tool for parallel performance profile analysis," Euro-Par 2003 Parallel Processing, pp. 17–26, 2003.

[22] S. Benedict, V. Petkov, and M. Gerndt, "PERISCOPE: An online-based distributed performance analysis tool," in Tools for High Performance Computing 2009, M. S. Müller, M. M. Resch, A. Schulz, and W. E. Nagel, Eds. Springer Berlin Heidelberg, 2010, pp. 1–16.

[23] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifiers (URI): generic syntax," 1998. [Online]. Available: http://www.hjp.at/doc/rfc/rfc3986.html

[24] J. C. Bezdek, R. Ehrlich, and W. Full, "FCM: The fuzzy c-means clustering algorithm," Computers & Geosciences, vol. 10, no. 2–3, pp. 191 – 203, 1984.

[25] C. Bischof, D. an Mey, and C. Iwainsky, "Brainware for green HPC," Computer Science - Research and Development, vol. 27, no. 4, pp. 227–233, 2012.

[26] H. Blockeel and J. Vanschoren, "Experiment databases: Towards an improved experimental methodology in machine learning," in PKDD, ser. Lecture Notes in Computer Science, vol. 4702. Springer, 2007, pp. 6–17.

[27] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (XML)," World Wide Web Journal, vol. 2, no. 4, pp. 27–66, 1997.

[28] P. Bridges, N. Doss, W. Gropp, E. Karrels, E. Lusk, and A. Skjellum, "User's Guide to MPICH, a Portable Implementation of MPI," Argonne National Laboratory, vol. 9700, pp. 60 439–4801, 1995.

[29] M. Brock, M. Proctor, B. McWhirter, and P. Lalloni, "MVEL Java-based expression language (version 2.1)," 2013. [Online]. Available: http://mvel.codehaus.org/

[30] P. Browne, JBoss Drools Business Rules. Packt Publishing Ltd, 2009.

[31] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, “A portable programming interface for performance evaluation on modern processors,” International Journal of High Performance Computing Applications, vol. 14, no. 3, pp. 189–204, 2000.

[32] B. Bruegge and A. Dutoit, Object-Oriented Software Engineering: Using Uml, Patterns, and Java. Prentice Hall PTR, 2010.

[33] B. Buck and J. K. Hollingsworth, “An API for runtime code patching,” International Journal of High Performance Computing Applications, vol. 14, no. 4, pp. 317–329, 2000.

[34] M. Burtscher, B.-D. Kim, J. Diamond, J. McCalpin, L. Koesterke, and J. Browne, “PerfExpert: An easy-to-use performance diagnosis tool for HPC applications,” in Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society, 2010, pp. 1–11.

[35] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro, “Choreography and orchestration: A synergic approach for system design,” in Service-Oriented Computing - ICSOC 2005, ser. Lecture Notes in Computer Science, B. Benatallah, F. Casati, and P. Traverso, Eds. Springer Berlin Heidelberg, 2005, vol. 3826, pp. 228–240.

[36] W. Carlson, J. Draper, D. Culler, K. Yelick, E. Brooks, and K. Warren, Introduction to UPC and language specification. Center for Computing Sciences, Institute for Defense Analyses, 1999.

[37] S. Chacon et al., “Git community book,” The Git Community, 2009.

[38] N. Chaimov, “Machine learning for automatic performance tuning,” University of Oregon Research Poster Contest 2012-13, 2012.

[39] B. Chapman, T. Curtis, S. Pophale, S. Poole, J. Kuehn, C. Koelbel, and L. Smith, “Introducing OpenSHMEM: SHMEM for the PGAS community,” in Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model. ACM, 2010, p. 2.

[40] C. Chen, J. Chame, and M. Hall, “CHiLL: A framework for composing high-level loop transformations,” University of Southern California, Tech. Rep, pp. 08–897, 2008.

[41] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, M. Ripeanu, B. Schwartzkopf et al., “Giggle: a framework for constructing scalable replica location services,” in Proceedings of the 2002 ACM/IEEE conference on Supercomputing. IEEE Computer Society Press, 2002, pp. 1–17.

[42] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana et al., "Web services description language (WSDL) 1.1," 2001.

[43] G. K. Christian Bauer and G. Gregory, Java Persistence with Hibernate, Second Edition, ser. Manning Pubs Co Series. Manning Publications Company, 2013.

[44] I. Chung, J. K. Hollingsworth et al., "Using information from prior runs to improve automated tuning systems," in Proceedings of the 2004 ACM/IEEE conference on Supercomputing. IEEE Computer Society, 2004, p. 30.

[45] R. Clint Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimizations of software and the ATLAS project," Parallel Computing, vol. 27, no. 1, pp. 3–35, 2001.

[46] M. Colgrove, C. Hansen, and K. Karavanic, "Managing parallel performance experiments with PPerfDB," in Parallel and Distributed Computing and Networks: Proceedings of the 23-rd IASTED International Multi-Conference on Applied Informatics, 2005.

[47] G. Cong, I.-H. Chung, H.-F. Wen, D. Klepacki, H. Murata, Y. Negishi, and T. Moriyama, "A systematic approach towards automated performance analysis and tuning," IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 3, pp. 426–435, 2012.

[48] M. Cooper, "Advanced bash-scripting guide," 2012. [Online]. Available: http://www.tldp.org/LDP/abs/html/index.html

[49] D. Crockford, "JSON," 2006.

[50] T. W. Curry et al., "Profiling and tracing dynamic library usage via interposition," in USENIX Summer 1994 Technical Conference, 1994, pp. 267–278.

[51] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," IEEE Computational Science & Engineering, vol. 5, no. 1, pp. 46–55, 1998.

[52] T. H. Davenport, Process innovation: reengineering work through information technology. Boston, Mass.: Harvard Business School Press, 1993.

[53] J. Davis, M. Goel, C. Hylands, B. Kienhuis, E. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie et al., "Ptolemy II: Heterogeneous concurrent modeling and design in Java," University of California, Berkeley, Tech. Rep. UCB/ERL M, vol. 99, 1999.

[54] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, "Pegasus: Mapping scientific workflows onto the grid," in Grid Computing. Springer, 2004, pp. 11–20.

[55] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good et al., "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," Scientific Programming, vol. 13, no. 3, pp. 219–237, 2005.

[56] J. des Rivieres and J. Wiegand, "Eclipse: A platform for integrating development tools," IBM Systems Journal, vol. 43, no. 2, pp. 371–383, 2004.

[57] J. Dongarra, S. for Industrial, and A. Mathematics, LINPACK users' guide. Society for Industrial and Applied Mathematics (SIAM), 1979.

[58] M. Dumas and A. H. Ter Hofstede, "UML activity diagrams as a workflow specification language," in UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools. Springer, 2001, pp. 76–90.

[59] D. Eschweiler, M. Wagner, M. Geimer, A. Knüpfer, W. E. Nagel, and F. Wolf, "Open trace format 2–the next generation of scalable trace formats and support libraries," in Proceedings of the International Conference on Parallel Computing (ParCo), Ghent, 2011.

[60] G. Flor, G. Manduchi, T. Fredian, J. Stillemnan, and K. Klare, "MDS-PIUS: A software system for fast control and data acquisition in fusion experiments," in Proceedings of the Seventh Conference on Computer Applications in Nuclear, Particle and Plasma Physics. IEEE, 1991, pp. 109–116.

[61] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," Artificial intelligence, vol. 19, no. 1, pp. 17–37, 1982.

[62] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in Network and parallel computing. Springer, 2005, pp. 2–13.

[63] I. Foster, J. Vockler, M. Wilde, and Y. Zhao, "Chimera: A virtual data system for representing, querying, and automating data derivation," in Proceedings of the 14th International Conference on Scientific and Statistical Database Management. IEEE, 2002, pp. 37–46.

[64] M. Frigo and S. G. Johnson, "FFTW: Fastest Fourier transform in the west," in Astrophysics Source Code Library, record ascl: 1201.015, vol. 1, 2012, p. 01015.

[65] W. Frings, F. Wolf, and V. Petkov, "Scalable massively parallel I/O to task-local files," in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. ACM, 2009, p. 17.

[66] G. Fursin, "Collective tuning initiative: automating and accelerating development and optimization of computing systems," in GCC Developers' Summit, 2009.

[67] G. Fursin, Y. Kashnikov, A. Memon, Z. Chamski, O. Temam, M. Namolaru, E. Yom-Tov, B. Mendelson, A. Zaks, E. Courtois, F. Bodin, P. Barnard, E. Ashton, E. Bonilla, J. Thomson, C. Williams, and M. O'Boyle, "Milepost GCC: Machine learning enabled self-tuning compiler," International Journal of Parallel Programming, vol. 39, pp. 296–327, 2011.

[68] G. Fursin and O. Temam, "Collective optimization: A practical collaborative approach," ACM Transactions on Architecture and Code Optimization (TACO), vol. 7, no. 4, p. 20, 2010.

[69] Z. W. Geem, J. H. Kim, and G. Loganathan, "A new heuristic optimization algorithm: harmony search," Simulation, vol. 76, no. 2, pp. 60–68, 2001.

[70] K. Geiger, inside ODBC. Microsoft Press, 1995.

[71] M. Geimer, F. Wolf, B. Wylie, E. Ábrahám, D. Becker, and B. Mohr, "The Scalasca performance toolset architecture," Concurrency and Computation: Practice and Experience, vol. 22, no. 6, pp. 702–719, 2010.

[72] M. Geimer, P. Saviankou, A. Strube, Z. Szebenyi, F. Wolf, and B. Wylie, "Further improving the scalability of the Scalasca toolset," in Applied Parallel and Scientific Computing, ser. Lecture Notes in Computer Science, K. Jónasson, Ed. Springer Berlin Heidelberg, 2012, vol. 7134, pp. 463–473.

[73] H. J. Genrich and K. Lautenbach, "System modelling with high-level petri nets," Theoretical computer science, vol. 13, no. 1, pp. 109–135, 1981.

[74] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: from process modeling to workflow automation infrastructure," Distributed and parallel Databases, vol. 3, no. 2, pp. 119–153, 1995.

[75] M. Gerndt, "Performance analysis tools," in Encyclopedia of Parallel Computing, D. Padua, Ed. Springer US, 2011, pp. 1515–1522.

[76] M. Gerndt and E. Kereku, "Search strategies for automatic performance analysis tools," Euro-Par 2007 Parallel Processing, pp. 129–138, 2007.

[77] M. Gerndt and S. Strohhäcker, "Distribution of Periscope analysis agents on ALTIX 4700," in Proceedings of the International Conference on the Parallel Computing (ParCo 2007). Advances in Parallel Computing, vol. 15, 2008, pp. 113–120.

[78] Google Inc., "Google gson: A java library to convert json to java objects and vice-versa." 2013. [Online]. Available: http://code.google.com/p/google-gson/

[79] Google Inc., "Protocol buffers: Google's data interchange format," 2013. [Online]. Available: http://code.google.com/p/protobuf/

[80] R. Gronback, Eclipse Modeling Project: A Domain-Specific Language Toolkit, ser. The Eclipse Series. Addison-Wesley, 2009.

[81] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," Parallel computing, vol. 22, no. 6, pp. 789–828, 1996.

[82] D. Grune et al., "Concurrent versions system, a method for independent cooperation," Report IR-114, Vrije University, Amsterdam, 1986.

[83] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," ACM SIGKDD Explorations Newsletter, vol. 11, no. 1, pp. 10–18, 2009.

[84] M. Hammer and J. Champy, Reengineering the corporation. Harper Business, 1988.

[85] M. Haneda, P. M. Knijnenburg, and H. A. Wijshoff, "Automatic selection of compiler options using non-parametric inferential statistics," in 14th International Conference on Parallel Architectures and Compilation Techniques (PACT). IEEE, 2005, pp. 123–132.

[86] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," Computer, vol. 41, no. 7, pp. 33–38, 2008.

[87] K. Huck and A. Malony, "PerfExplorer: A performance data mining framework for large-scale parallel computing," in Proceedings of the ACM/IEEE Conference on Supercomputing (SC 2005)., nov. 2005, p. 41.

[88] K. A. Huck, A. D. Malony, R. Bell, and A. Morris, "Design and implementation of a parallel performance data management framework," in Parallel Processing, 2005. ICPP 2005. International Conference on. IEEE, 2005, pp. 473–482.

[89] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden et al., "The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models," Bioinformatics, vol. 19, no. 4, pp. 524–531, 2003.

[90] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services," Nucleic acids research, vol. 34, no. suppl 2, pp. W729–W732, 2006.

[91] IBM, "Common Public License (CPL)," May 2001. [Online]. Available: http://opensource.org/licenses/cpl1.0.php

[92] IBM, "Dramatically improve the way work gets done with IBM Business Process Manager," IBM Corporation, Thought Leadership White Paper, 2012.

[93] IBM, "IBM Business Process Manager (version 7.5)," 2013. [Online]. Available: http://www.ibm.com/software/integration/business-process-manager

[94] IBM, "IBM Tivoli Software," 2013. [Online]. Available: http://www.ibm.com/software/tivoli/

[95] IBM, "InfoSphere Master Data Management (version 10.1)," 2013. [Online]. Available: http://www.ibm.com/software/data/infosphere/mdm/

[96] IEEE Standards Board, "IEEE guide for developing system requirements specifications," IEEE Std 1233, 1998 Edition, pp. 1–36, 1998.

[97] IEEE Standards Board, "IEEE guide for information technology - system definition - concept of operations (ConOps) document," IEEE Std 1362-1998, pp. 1–24, 1998.

[98] IEEE Standards Board, "IEEE recommended practice for software requirements specifications," IEEE Std 830-1998, pp. 1–40, 1998.

[99] IEEE Standards Board, "IEEE standard for conceptual modeling language syntax and semantics for IDEF1X/Sub 97/ (IDEF/Sub Object)," IEEE Std 1320.2-1998, pp. i–, 1998.

[100] IEEE Standards Board, "IEEE standard for functional modeling language - syntax and semantics for IDEF0," IEEE Std 1320.1-1998, pp. i–, 1998.

[101] IEEE Standards Board, "IEEE standard for software maintenance," IEEE Std 1219-1998, 1998.

[102] IEEE Standards Board, "IEEE standard for software reviews," IEEE Std 1028-1997, pp. i–37, 1998.

[103] IEEE/ANSI Standards Organizations, "IEEE standard for software unit testing," ANSI/IEEE Std 1008-1987, 1986.

[104] IEEE/ISO/IEC Standards Organizations, “International standard - iso/iec 14764 IEEE std 14764-2006 software engineering, software life cycle processes and maintenance,” ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998, 2006.

[105] IEEE/ISO/IEC Standards Organizations, “Systems and software engineering – life cycle processes – requirements engineering,” ISO/IEC/IEEE 29148:2011(E), pp. 1–94, 2011.

[106] Intel, MKL Team, “Intel math kernel library,” 2012. [Online]. Available: http://software.intel.com/en-us/intel-mkl

[107] C. Iwainsky, R. Altenfeld, D. Mey, and C. Bischof, “Enhancing brainware productivity through a performance tuning workflow,” in Euro-Par 2011: Parallel Processing Workshops, ser. Lecture Notes in Computer Science, M. Alexander, P. D’Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. Scott, J. Traff, G. Vallée, and J. Weidendorfer, Eds. Springer Berlin Heidelberg, 2012, vol. 7156, pp. 198–207.

[108] H. Jin and R. F. Van der Wijngaart, “Performance characteristics of the multi-zone NAS parallel benchmarks,” Journal of Parallel and Distributed Computing, vol. 66, no. 5, pp. 674–685, 2006.

[109] X. Jin and J. Han, “K-means clustering,” in Encyclopedia of Machine Learning. Springer, 2010, pp. 563–564.

[110] D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland et al., “Web services business process execution language version 2.0,” OASIS Standard, vol. 11, 2007.

[111] M. Jugel and S. Schmidt, “Snipsnap: the easy weblog and wiki software,” 2006.

[112] S. Kannan, M. Roberts, P. Mayes, D. Brelsford, and J. F. Skovira, “Workload management with LoadLeveler,” IBM Redbooks, p. 222, 2001.

[113] K. L. Karavanic, J. May, K. Mohror, B. Miller, K. Huck, R. Knapp, and B. Pugh, “Integrating database technology with comparison-based parallel performance diagnosis: The PerfTrack performance experiment management tool,” in Proceedings of the ACM/IEEE Supercomputing Conference (SC 2005). IEEE, 2005, pp. 39–39.

[114] K. Kawaguchi, S. Vajjhala, and J. Fialli, “The Java Architecture for XML Binding (JAXB) 2.1,” Sun Microsystems, Inc, 2006.

[115] G. Keller, A.-W. Scheer, and M. Nüttgens, Semantische Prozeßmodellierung auf der Grundlage" Ereignisgesteuerter Prozeßketten (EPK)". Inst. für Wirtschaftsinformatik, 1992.

[116] E. Kereku and M. Gerndt, "The monitoring request interface (mri)," in Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International. IEEE, 2006, pp. 8–pp.

[117] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic, "Ws-bpel extension for people–bpel4people," Joint white paper, IBM and SAP, vol. 183, p. 184, 2005.

[118] R. L. Knapp, K. Mohror, A. Amauba, K. L. Karavanic, A. Neben, T. Conerly, and J. May, "PerfTrack: Scalable application performance diagnosis for linux clusters," in 8th LCI International Conference on High-Performance Clustered Computing. Citeseer, 2007, pp. 15–17.

[119] A. Knüpfer, R. Brendel, H. Brunst, H. Mix, and W. Nagel, "Introducing the open trace format (OTF)," Computational Science–ICCS 2006, pp. 526–533, 2006.

[120] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. Müller, and W. Nagel, "The Vampir performance analysis tool-set," in Tools for High Performance Computing, M. Resch, R. Keller, V. Himmler, B. Krammer, and A. Schulz, Eds. Springer Berlin Heidelberg, 2008, pp. 139–155.

[121] A. Knüpfer, C. Rössel, D. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf, "Score-P: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir," in Tools for High Performance Computing 2011, H. Brunst, M. S. Müller, W. E. Nagel, and M. M. Resch, Eds. Springer Berlin Heidelberg, 2012, pp. 79–91.

[122] R. K. L. Ko, "A computer scientist's introductory guide to business process management (bpm)," Crossroads, vol. 15, no. 4, pp. 4:11–4:18, 2009.

[123] D. Köhn and N. Le Novere, "SED-ML – an XML format for the implementation of the MIASE guidelines," in Computational Methods in Systems Biology. Springer, 2008, pp. 176–190.

[124] P. Kruchten, The rational unified process: an introduction. Addison-Wesley Professional, 2004.

[125] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for fortran usage," ACM Transactions on Mathematical Software (TOMS), vol. 5, no. 3, pp. 308–323, 1979.

[126] P. J. Leach, M. Mealling, and R. Salz, "A universally unique identifier (UUID) URN namespace," 2005. [Online]. Available: http://tools.ietf.org/html/rfc4122.html/

[127] H. Leather, E. Bonilla, and M. O'Boyle, "Automatic feature generation for machine learning based optimizing compilation," in International Symposium on Code Generation and Optimization (CGO 2009)., 2009, pp. 81–91.

[128] E. A. Lee and S. Neuendorffer, MoML: A Modeling Markup Language in SML: Version 0.4. Citeseer, 2000.

[129] B. Leuf and W. Cunningham, The Wiki Way: Quick Collaboration on the Web. Addison-Wesley, 2001.

[130] Leuven University & Leiden University, "OpenML - Open Science Platform fir Machine Learning." [Online]. Available: http://expdb.cs.kuleuven.be/

[131] K. A. Lindlan, J. Cuny, A. D. Malony, S. Shende, B. Mohr, R. Rivenburgh, and C. Rasmussen, "A tool framework for static and dynamic analysis of object-oriented software with templates," in Supercomputing, ACM/IEEE 2000 Conference. IEEE, 2000, pp. 49–49.

[132] J. Long, ITIL Version 3 at a Glance: Information Quick Reference. Springer London, Limited, 2008.

[133] D. C. Luckham, The power of events. Addison-Wesley Reading, 2002, vol. 204.

[134] D. C. Luckham, Event Processing for Business: Organizing the Real-time Enterprise. Wiley, 2011.

[135] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," Concurrency and Computation: Practice and Experience, vol. 18, no. 10, pp. 1039–1065, 2006.

[136] B. Ludäscher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D. D. Roure, J. Freire, C. Goble, M. Jones et al., "Scientific process automation and workflow management," Scientific Data Management: Challenges, Existing Technology, and Deployment, Computational Science Series, pp. 476–508, 2009.

[137] A. Malony, S. Shende, W. Spear, C. Lee, and S. Biersdorff, "Advances in the TAU performance system," in Tools for High Performance Computing 2011, H. Brunst, M. S. Müller, W. E. Nagel, and M. M. Resch, Eds. Springer Berlin Heidelberg, 2012, pp. 119–130.

[138] G. Manduchi and G. Flor, "Linda: The experiment description language of the mds-plus data acquisition system," Review of Scientific Instruments, vol. 61, no. 10, pp. 3292–3294, 1990.

[139] MATLAB, MATLAB and Statistics Toolbox Release 2012b. Natick, Massachusetts: The MathWorks Inc., 2013.

[140] Apache Software Foundation, "Apache hadoop," 2012. [Online]. Available: http://hadoop.apache.org/

[141] CAPS Enterprise, Cray Inc., NVIDIA and the Portland Group, "The OpenACC Application Programming Interface," Online, 2011. [Online]. Available: http://www.openacc-standard.org/

[142] Nvidia CUDA, "NVIDIA CUDA programming guide," 2011.

[143] Workflow Management Coalition, "Process definition interface - XML process definition language," Workflow Management Coalition, Tech. Rep. WFMC-TC-1025, 2008, document Number WFMC-TC-1025.

[144] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, pp. 19–25, 1995.

[145] Message Passing Interface Forum, "Mpi: A message-passing interface standard, version 2.2," MPIF, Specification, September 2009. [Online]. Available: http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf

[146] D. Mey, S. Biersdorf, C. Bischof, K. Diethelm, D. Eschweiler, M. Gerndt, A. Knüpfer, D. Lorenz, A. Malony, W. Nagel, Y. Oleynik, C. Rössel, P. Saviankou, D. Schmidl, S. Shende, M. Wagner, B. Wesarg, and F. Wolf, "Score-P: A unified performance measurement system for petascale applications," in Competence in High Performance Computing 2010, C. Bischof, H.-G. Hegering, W. E. Nagel, and G. Wittum, Eds. Springer Berlin Heidelberg, 2012, pp. 85–97.

[147] R. Miceli, G. Civario, A. Sikora, E. César, M. Gerndt, H. Haitof, C. Navarrete, S. Benkner, M. Sandrieser, L. Morin, and F. Bodin, "Autotune: A plugin-driven approach to the automatic tuning of parallel applications," in Applied Parallel and Scientific Computing, ser. Lecture Notes in Computer Science, P. Manninen and P. Öster, Eds. Springer Berlin Heidelberg, 2013, vol. 7782, pp. 328–342.

[148] J. Michalakes, S. Chen, J. Dudhia, L. Hart, J. Klemp, J. Middlecoff, and W. Skamarock, "Development of a next generation regional weather research and forecast model," in Developments in Teracomputing: Proceedings of the Ninth ECMWF Workshop on the use of high performance computing in meteorology. World Scientific, 2001, pp. 269–276.

[149] A. K. Miller, J. Marsh, A. Reeve, A. Garny, R. Britten, M. Halstead, J. Cooper, D. P. Nickerson, and P. F. Nielsen, "An overview of the CellML API and its implementation," BMC bioinformatics, vol. 11, no. 1, p. 178, 2010.

[150] P. Missier, K. Belhajjame, J. Zhao, M. Roos, and C. Goble, "Data lineage model for Taverna workflows with lightweight annotation requirements," in Provenance and Annotation of Data and Processes. Springer, 2008, pp. 17–30.

[151] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble, "Taverna, reloaded," in Scientific and Statistical Database Management. Springer, 2010, pp. 471–481.

[152] M. S. Müller, M. van Waveren, R. Lieberman, B. Whitney, H. Saito, K. Kumaran, J. Baron, W. C. Brantley, C. Parrott, T. Elken et al., "SPEC MPI2007 – an application benchmark suite for parallel systems using MPI," Concurrency and Computation: Practice and Experience, vol. 22, no. 2, pp. 191–205, 2010. [Online]. Available: http://www.spec.org/mpi/

[153] T. Müller, "H2 database engine," 2013. [Online]. Available: http://www.h2database.com

[154] A. Munshi et al., "The opencl specification," Khronos OpenCL Working Group, vol. 1, pp. l1–15, 2009.

[155] T. Murata, "Petri nets: Properties, analysis and applications," Proceedings of the IEEE, vol. 77, no. 4, pp. 541–580, 1989.

[156] Y. L. Nelson, B. Bansal, M. Hall, A. Nakano, and K. Lerman, "Model-guided performance tuning of parameter values: A case study with molecular dynamics visualization," in IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008). IEEE, 2008, pp. 1–8.

[157] Object Management Group (OMG), "Business Process Model and Notation (BPMN) Version 2.0," OMG Specification, Object Management Group, 2011.

[158] D. Oracle Berkeley, "Java edition," Available Online. URL: http://www. oracle. com/database/berkeley-db/je/index. html, 2013.

[159] B. O'sullivan, "Making sense of revision-control systems," Communications of the ACM, vol. 52, no. 9, pp. 56–62, 2009.

[160] B. O'Sullivan, Mercurial: The definitive guide. O'Reilly Media, 2009.

[161] B. Pagés, "BOUML – a free uml toolbox (ver. 6.4)," 2013. [Online]. Available: http://www.bouml.fr/

[162] S. R. Palmer and M. Felsing, A practical guide to feature-driven development. Pearson Education, 2001.

[163] Z. Pan and R. Eigenmann, "Fast and effective orchestration of compiler optimizations for automatic performance tuning," in International Symposium on Code Generation and Optimization (CGO 2006). IEEE, 2006.

[164] Z. Pan and R. Eigenmann, "Peak – a fast and effective performance tuning system via compiler optimization orchestration," ACM Trans. Program. Lang. Syst., vol. 30, no. 3, 2008.

[165] C. Pancake, "Applying human factors to the design of performance tools," in Euro-Par'99 Parallel Processing, ser. Lecture Notes in Computer Science, P. Amestoy, P. Berger, M. Daydé, D. Ruiz, I. Duff, V. Frayssé, and L. Giraud, Eds. Springer Berlin Heidelberg, 1999, vol. 1685, pp. 44–60.

[166] V. Petkov and M. Gerndt, "Integrating parallel application development with performance analysis in Periscope," in IPDPS Workshops. IEEE, 2010, pp. 1–8.

[167] S. L. Pfleeger and J. M. Atlee, Software Engineering: Theory and Practice (4. ed.). Prentice Hall, 2010.

[168] C. Pilato, B. Collins-Sussman, and B. Fitzpatrick, Version Control with Subversion, ser. Oreilly Series. O'Reilly Media, 2009.

[169] R. Prodan and T. Fahringer, "Zenturio: An experiment management system for cluster and grid computing," in Proceedings of the IEEE International Conference on Cluster Computing. IEEE, 2002, pp. 9–18.

[170] M. Püschel, J. M. Moura, B. Singer, J. Xiong, J. Johnson, D. Padua, M. Veloso, and R. W. Johnson, "Spiral: A generator for platform-adapted libraries of signal processing alogorithms," International Journal of High Performance Computing Applications, vol. 18, no. 1, pp. 21–45, 2004.

[171] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar, "OMF: a control and management framework for networking testbeds," SIGOPS Oper. Syst. Rev., vol. 43, no. 4, pp. 54–59, Jan. 2010.

[172] R. E. Raygan and D. G. Green, "Internet collaboration: Twiki," in SoutheastCon, 2002. Proceedings IEEE. IEEE, 2002, pp. 137–141.

[173] G. Reese, Database Programming with JDBC and JAVA. O'Reilly Media, Inc., 2000.

[174] R. Reussner, S. Becker, J. Happe, H. Koziolek, K. Krogmann, and M. Kuperberg, "The palladio component model," Interner Bericht, vol. 21, 2007.

[175] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed, "Autopilot: Adaptive control of distributed applications," in Proceedings of the Seventh International Symposium on High Performance Distributed Computing. IEEE, 1998, pp. 172–179.

[176] S. Robertson and J. Robertson, Mastering the Requirements Process: Getting Requirements Right. Pearson Education, 2012.

[177] W. Royse, "Managing the development of large software systems: concepts and techniques," in IEEE WESTCON, 1970.

[178] A. Rubin, Statistics for Evidence-Based Practice and Evaluation, ser. Research, Statistics, and Program Evaluation Series. BROOKS COLE Publishing Company, 2012.

[179] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual. Pearson Higher Education, 2004.

[180] M. Salatino, Jbpm 5 Developer Guide. Packt Publishing, Limited, 2012.

[181] SAP, "SAP Business Management Software." [Online]. Available: http://www.sap.com

[182] A. W. Scheer, ARIS: business process modeling. Springer, 2000.

[183] K. Schwaber and M. Beedle, Agile software development with Scrum. Prentice Hall Upper Saddle River, 2002, vol. 1.

[184] Score-P Consortium, Score-P 1.1.x User Manual, 2012. [Online]. Available: http://www.score-p.org

[185] C. Seragiotto, H.-L. Truong, T. Fahringer, B. Mohr, M. Gerndt, and T. Li, "Standardized intermediate representation for Fortran, Java, C and C++ Programs," APART Working Group Technical Report, Institute for Software Science, University of Vienna, Octorber, 2004.

[186] S. Shende and A. D. Malony, "The TAU parallel performance system," International Journal of High Performance Computing Applications, vol. 20, no. 2, pp. 287–311, 2006.

[187] S. Shende, A. Malony, and A. Morris, "Improving the scalability of performance evaluation tools," in Applied Parallel and Scientific Computing, ser. Lecture Notes in Computer Science, K. Jónasson, Ed. Springer Berlin Heidelberg, 2012, vol. 7134, pp. 441–451.

[188] J. Sinur, W. R. Schulte, J. B. Hill, and T. Jones, "Magic quadrant for intelligent business process management suites," Gartner Inc., Tech. Rep. G00224913, 2012.

[189] F. Song, F. Wolf, N. Bhatia, J. Dongarra, and S. Moore, "An algebra for cross-experiment performance analysis," in ICPP. IEEE Computer Society, 2004, pp. 63–72.

[190] O. A. Sopeju, M. Burtscher, A. Rane, and J. Browne, "Autoscope: Automatic suggestions for code optimizations using perfexpert," Evaluation, 2011.

[191] Sparx Systems, "Enterprise architect 10," 2013. [Online]. Available: www.sparxsystems.com

[192] SPEC Benchmarks, "Standard performance evaluation corporation (SPEC)," 2013. [Online]. Available: http://www.spec.org/

[193] J. Spolsky, "Hg init: a mercurial tutorial," 2011.

[194] T. S. Staines, "Intuitive mapping of uml 2 activity diagrams into fundamental modeling concept petri net diagrams and colored petri nets," in Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the. IEEE, 2008, pp. 191–200.

[195] V. Taylor, X. Wu, and R. Stevens, "Prophesy: an infrastructure for performance analysis and modeling of parallel and grid applications," SIGMETRICS Perform. Eval. Rev., vol. 30, no. 4, pp. 13–18, 2003.

[196] The HSQL Development Group, "HyperSQL Database Engine (HSQLDB)," 2013. [Online]. Available: http://www.hsqldb.org/

[197] The JBoss Drools team, "Drools Expert - Declarative Business Rule Engine." [Online]. Available: http://www.jboss.org/drools/drools-expert.html

[198] The JBoss Drools team, "Drools Fusion - Complex Event Processing." [Online]. Available: http://www.jboss.org/drools/drools-fusion.html

[199] The JBoss Drools team, "Drools Guvnor - Business Rules Manager." [Online]. Available: http://www.jboss.org/drools/drools-guvnor

[200] The JBoss Drools team, "jBPM 5 - Workflow Automation Engine." [Online]. Available: http://www.jboss.org/jbpm

[201] The jBPM Team, "jbpm user guide: Persistence and transactions." [Online]. Available: http://docs.jboss.org/jbpm/v5.4/userguide/ch.core-persistence.html

[202] The OptaPlanner team, "OptaPlanner - Automated Process Planning Engine." [Online]. Available: http://www.optaplanner.org/

[203] A. Tiwari, C. Chen, J. Chame, M. Hall, and J. Hollingsworth, "A scalable auto-tuning framework for compiler optimization," in IEEE International Symposium on Parallel Distributed Processing (IPDPS 2009)., 2009, pp. 1–12.

[204] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures, San Diego CA, 2010.

[205] S. Triantafyllis, M. Vachharajani, N. Vachharajani, and D. I. August, "Compiler optimization-space exploration," in Code Generation and Optimization, 2003. CGO 2003. International Symposium on. IEEE, 2003, pp. 204–215.

[206] H.-L. Truong and T. Fahringer, "Scalea: a performance analysis tool for parallel programs," Concurrency and Computation: Practice and Experience, vol. 15, no. 11-12, pp. 1001–1025, 2003.

[207] E. Tsang, Foundations of constraint satisfaction. Academic press London, 1993, vol. 289.

[208] D. Turi, P. Missier, C. Goble, D. De Roure, and T. Oinn, "Taverna workflows: Syntax and semantics," in e-Science and Grid Computing, IEEE International Conference on. IEEE, 2007, pp. 441–448.

[209] W. M. van der Aalst, "The application of petri nets to workflow management," Journal of circuits, systems, and computers, vol. 8, no. 01, pp. 21–66, 1998.

[210] W. M. van der Aalst, "Business process management: A personal view," Business Process Management Journal, vol. 10, no. 2, pp. 135–139, 2004.

[211] W. M. Van Der Aalst, A. H. Ter Hofstede, and M. Weske, “Business process management: A survey,” in Business process management. Springer, 2003, pp. 1–12.

[212] VI-HPS, “Virtual Institute - High Productivity Supercomputing (VI-HPS),” 2013. [Online]. Available: http://www.vi-hps.org/

[213] Virtual Institute - High Productivity Supercomputing (VI-HPS), “Performance analysis workflow using Score-P,” 2013. [Online]. Available: https://silc.zih.tu-dresden.de/scorep-current/html/workflow.html

[214] Visual Paradigm International, “Visual paradigm for UML (ver. 10.1),” 2013. [Online]. Available: http://www.visual-paradigm.com/product/vpuml/

[215] D. Viswanathan and S. Liang, “Java virtual machine profiler interface,” IBM Systems Journal, vol. 39, no. 1, pp. 82–95, 2000.

[216] R. Vuduc, J. W. Demmel, and K. A. Yelick, “Oski: A library of automatically tuned sparse matrix kernels,” in Journal of Physics: Conference Series, vol. 16, no. 1. IOP Publishing, 2005, p. 521.

[217] D. Waltemath, R. Adams, D. A. Beard, F. T. Bergmann, U. S. Bhalla, R. Britten, V. Chelliah, M. T. Cooling, J. Cooper, E. J. Crampin, A. Garny, S. Hoops, M. Hucka, P. Hunter, E. Klipp, C. Laibe, A. K. Miller, I. Moraru, D. Nickerson, P. Nielsen, M. Nikolski, S. Sahle, H. M. Sauro, H. Schmidt, J. L. Snoep, D. Tolle, O. Wolkenhauer, and N. Le Novère, “Minimum information about a simulation experiment (miase),” PLoS Comput Biol, vol. 7, no. 4, p. e1001122, 04 2011.

[218] D. Waltemath, R. Adams, F. T. Bergmann, M. Hucka, F. Kolpakov, A. K. Miller, I. I. Moraru, D. Nickerson, S. Sahle, J. L. Snoep et al., “Reproducible computational biology experiments with SED-ML – the simulation experiment description markup language,” BMC systems biology, vol. 5, no. 1, p. 198, 2011.

[219] C. Walton, “Eclipse wiki editor plugin (ver. 2.7),” 2013. [Online]. Available: http://eclipsewiki.sourceforge.net/

[220] H.-Q. Wang, “Research and application of semi-supervised cluster by harmony search,” Chinese Core Journal of Computer Engineering and Design, vol. 33, no. 7, pp. 2797–2803, 2012.

[221] H. Wang, M. Gerndt, and V. Petkov, "Research and application of harmony search fuzzy clustering with feature selection," Chinese Core Journal of Computer Engineering and Applications, October 2013, to be published.

[222] G. Watson, C. Rasmussen, and B. Tibbitts, "An integrated approach to improving the parallel application development process," in IEEE International Symposium on Parallel Distributed Processing (IPDPS 2009), 2009, pp. 1–8.

[223] M. Weske, Business Process Modelling Foundation. Springer Berlin Heidelberg, 2012.

[224] S. A. White, "Introduction to BPMN," IBM Cooperation, pp. 28–29, 2004.

[225] M. V. Wilkes, "The memory gap and the future of high performance memories," ACM SIGARCH Computer Architecture News, vol. 29, no. 1, pp. 2–7, 2001.

[226] L. Wingerd, Practical Perforce, ser. Practical Series. O'Reilly Media, 2007.

[227] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in Job Scheduling Strategies for Parallel Processing. Springer, 2003, pp. 44–60.