THE UNIVERSITY of ADELAIDE

Deployment Optimisation Insights for Evolving Software Systems within a Constrained Computing Environment

by

Gavin Puddy

Thesis submitted for the degree of Doctor of Philosophy

Supervised by Prof. Katrina Falkner
and A/Prof. Claudia Szabo

The University of Adelaide
School of Computer Science
Faculty of Engineering, Computer
and Mathematical Sciences

2021

I am dedicating my PhD thesis to my two boys, Jackson and Mitchell, and my loving wife, Mel. I thank you for your support, care and patience over the journey.

Acknowledgments

I would like to acknowledge the support and patience from my supervisors Professor Katrina Falkner and A/Professor Claudia Szabo.

I would also like to acknowledge the support given to me from members of the DIG research team.

Abstract

For the successful integration of System of Systems (SoS) type designs, there is a critical need to understand the non-functional aspects of the design and how these design aspects are affected by new or evolved operational conditions and subsequent deployment scenarios. An undersea sensor network system is one example of an SoS design where non-functional design aspects, such as power consumption and temporal performance, are critical to the overall system performance and capability, and are largely influenced by high-level operational needs.

There is a crucial need to develop methodologies for modelling and understanding non-functional properties and corresponding deployment scenarios early in the development and integration cycle. This includes the initial development and subsequent evolution upgrade cycles.

This thesis presents literature that shows there are a small number of pockets of research looking into measurement-based techniques for performance prediction of SoS designs early in the development cycle. Furthermore, the literature shows that

System Execution Modelling (SEM) techniques lead the way in providing low-level insight into non-functional requirement behaviours and performances. However, the techniques do not have sufficient capability for managing the complexities and scale of the deployment of the software that make up SoS designs.

The literature also shows that there are optimisation techniques for problem solving for many computing paradigms, including software deployment. However, these approaches are not adequate for software component deployment. Furthermore, the literature shows system evolution performance prediction techniques are either postmortem-based or largely abstract in nature, and not to the level of fidelity required for SoS performance insights.

This thesis details new research to address the capability gap for modelling deployment of large-scale evolving SoS. It will introduce a new software deployment optimisation technique and modelling language that allows for evolution characteristic definition and construction of evolved designs. It will present a software deployment optimisation technique, algorithms and frameworks, and a capability that enables software deployment to be shaped by high-level requirements.

We demonstrate the new optimisation capability with an undersea sensor system case study, where we explore design and deployment options for achieving high-level defined performances.

The testing and analysis presented in thesis shows the new optimisation approach operates and predicts correctly for each element of its optimisation algorithm. While some performance shortfalls exist, largely due to the model fidelity improvement requirements, our extensive verification and validation indicates the

correct overall performance and ability to identify software component deployment options in response to high-level requirements.

To complement the new optimisation capability, this thesis also introduces a new DSML to allow for exploration of software deployment optimisation for evolving systems. Known as CEML, this new language allows system designers to gain insight into how best to utilise its available computing resources when the software system evolves. The application of this new language was demonstrated by working through an undersea sensor system evolution case study.

This thesis introduces four contributions to the research community associated with modelling and predicting system design performance, including system evolution.

Declaration of Authorship

I certify that this work contains no material that has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I acknowledge that copyright of published works contained within this thesis resides with the copyright holder(s) of those works.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the

University to restrict access for a period of time.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Gavin Stanley Puddy

October 2021

Publications

G. Puddy, "Performance and Integration Evaluation for Combat Systems (PIECS) Modelling and Analysis Environment: An approach to support reducing combat system design risks," 2019 Military Communications and Information Systems Conference (MilCIS), 2019, pp. 1-6, doi: 10.1109/MilCIS.2019.8930733.

M. Britton, D. Ranathunga, A. Coyle and G. Puddy, "Modelling Computational Workload in a Maritime Combat System," 2019 Military Communications and Information Systems Conference (MilCIS), 2019, pp. 1-6, doi: 10.1109/MilCIS.2019.8930764.

M. Stewart, M. Britton, D. Ranathunga and G. Puddy, "Workload Models to Evaluate Heterogeneous Compute Environments Supporting Combat Systems," 2019 Military Communications and Information Systems Conference (MilCIS), 2019, pp. 1-6, doi: 10.1109/MilCIS.2019.8930793.

Falkner, K., Szabo, C., Chiprianov, V., Puddy, G., Rieckmann, M., Fraser, D., & Aston, C. (2018). Model-driven performance prediction of systems of systems. Software and Systems Modeling, 17(2), 415-441, https://doi.org/10.1007/s10270-016-0547-8

Falkner, K., Chiprianov, V., Falkner, N., Szabo, C., & Puddy, G. (2014). A Model-Driven Engineering Method for DRE Defense Systems Performance Analysis and Prediction. In A. Bagnato, L. Indrusiak, I. Quadri, & M. Rossi (Ed.), Handbook of Research on Embedded Systems Design (pp. 301-326). IGI Global. http://doi:10.4018/978-1-4666-6194-3.ch012

Chiprianov, V., Falkner, K., Szabo, C., & Puddy, G. (2014). Architectural support for model-driven performance prediction of distributed real-time embedded systems of systems. In P. Avgeriou, & U. Zdun (Eds.), Software architecture: 8th European Conference, ECSA 2014 Vienna, Austria, August 25-29, 2014: proceedings Vol. 8627 LNCS (pp. 357-364). Vienna, Austria: Springer International Publishing.

Britton, M., Falkner, K., & Puddy, G. (2013). Combat management systems: Predicting performance early in the design lifecycle. In Proceedings of the 2013 Military Communications and Information Systems Conference, MilCIS 2013 (pp. 1-6). United States: IEEE, doi: 10.1109/MilCIS.2013.6694492.

Falkner, K., Chiprianov, V., Falkner, N., Szabo, C., & Puddy, G. (2013). Modeling scenarios for the performance prediction of distributed real-time embedded systems. In 2013 Military Communications and Information Systems Conference (MilCIS) (pp. 1-6). United States: IEEE, doi: 10.1109/MilCIS.2013.6694495.

Falkner, K., Chiprianov, V., Falkner, N., Szabo, C., Hill, J., Puddy, G., . . . Wallis, A. (2013). Model-driven performance prediction of distributed real-time embedded defence systems. In Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS (pp. 155-158). USA: IEEE, doi: 10.1109/ICECCS.2013.29.

Contents

List of Tables

List of Figures

1. Introduction

1.1 Constrained System Design

Systems of systems (SoS) are collections of systems that work together to achieve a common purpose, where each individual system is typically independent in its operation and management. SoS design approaches are found across many domains including defence (Manthorpe, 1996), power networks (Perez et al., 2013), space (NASA, 2005) and avionics and automotive systems (Farcas et al., 2010). SoS are characterised by emergent behaviours resulting from the integration of independently operating and managed heterogeneous systems that are typically geographically distributed (Maier, 1998; Sage and Cuppan, 2001). In addition, SoS are never viewed as achieving a finalised state, but instead are required to evolve in response to user requirements or operational environment condition changes (Cook, 2001).

SoS such as defence command and control systems and other mission-critical distributed real-time and embedded (DRE) systems, can have life cycles in the decades (Merola, 2006) and will continue to evolve throughout that time as individ-

ual systems are updated or replaced. Moreover, mission-critical DRE systems are rarely built from scratch, and integrate legacy systems (Weiderman et al., 1997). This therefore imposes limitations for integration options and methods for operational deployment, resulting in unexpected and potentially undesirable overall system performances.

While an understanding of both functional and non-functional aspects of the SoS architecture is important, non-functional aspects are of greater concern for resource constrained host platforms, as there is little to no ability to improve processing capabilities once developed. With strict budget allocations for space, weight and power (SWaP) for various systems installed on many of these platforms, any early insight into the performance of the SoS from its corresponding deployment becomes crucial in preventing an imbalance between resource availability and demand, resulting in performance shortfalls and potentially expensive rebuilds.

The successful integration of systems within a SoS context has been identified as one of the most substantial challenges facing military systems development (Kewley et al., 2008). This challenge involves interfacing systems appropriately to enable the desired system interactions to promote overall capability to satisfy mission requirements. This is a challenge that is also a key concern within the aerospace domain, and is becoming increasingly important in other domains such as health and energy (Madni and Sievers, 2014).

Accordingly, there is a critical need to understand the non-functional aspects (such as quality of service, power, size, timing) of the SoS integration and to explore how they are affected by changes (static or dynamic) to new or evolved

operational conditions, and subsequent deployment. An examples of this could be a remotely deployed distributed sensor network SoS requiring upgrades to certain systems, as well as the need to improved battery power supply duration times for identified parts of the SoS.

As a result, there is a need to understand whether system upgrades are going to firstly produce an increased resource footprint and whether the underlying computing infrastructure (constrained) can support those new resource demands. Then, there is the question of whether or not the upgrade and resource demand level change will impact temporal performance. Lastly, there are questions about the impact any changes will have on battery power supply capabilities. In regards to addressing power supply duration improvements, insight would be needed to understand the impact of reducing the power supply levels (leading to reduced levels of processing capabilities), such as whether or not the computing resource demands can still be satisfied with this reduction in processing capabilities, or if there is a need to explore different deployment options to ensure the requirements remain satisfied.

Another challenge for SoS development is to minimise any risk of architectural change late in the development cycle, and the associated high costs thereof. Development of modelling methodologies is seen as crucial in trying to prevent late changes by gaining insight into deployment and non-functional properties early in the development and integration cycle (Balsamo et al., 2004). Such a benefit would also be present when using modelling methods to inform an understanding of the impacts of emergent behaviours for evolving SoS when deployed under

constrained SWaP budgets.

This early insight cannot be achieved with traditional software engineering methodologies, as they focus primarily on functional aspects of the system design and leave the non-functional aspects until the final phases of the development life cycle, when components have been integrated, deployed and configuration tested (Balsamo et al., 2004; Happe, Koziolek, and Reussner, 2011; Jasmine and Vasantha, 2007). While functional requirements are always a critical focus, a methodology is required to allow for modelling non-functional requirements at early design stages. This would allow for evaluation of satisfaction of non-functional requirements associated with real computing platforms, explore deployment and integration constraints and permit in-depth trade-off analysis with significant benefits (Nawinna and Sandeepani, 2020; Martens, 2011).

An undersea sensor network system is one example of a constrained SoS. This type of SoS may consist of a variable number of sensor nodes, either fixed to the seabed or mobile, and deployed to cover a designated investigation area. The SoS can perform data collection operations, storage and forwarding operations, and routing to a central collection node (Jindal, Saxena, and Singh, 2014 & Ovaliadis, Savage, and Kanakaris, 2010). Undersea sensor network systems have application in many domains such as mining exploration, disaster monitoring for floods, tsunamis, earthquakes, volcano eruptions and oil spills, along with military use in such areas as underwater mine detection and submarine surveillance, assisted navigation, and sports (Felemban et al., 2015).

Many challenges must be addressed by an undersea sensor network system

design and deployment, not only because of the acoustic and resource constraint context, but also because of the complexities resulting from environmental impacts with having to deploy into the harsh undersea environment. The SoS design and deployment has to account for non-functional attributes such as limited bandwidth, communication power requirements, temporal performance impact from long and variable propagation delays, significant error rates, computational power demands, power constraints, resource utilisation and system failures due to environmental characteristics such as corrosion and fouling (Akyildiz, Pompili, and Melodia, 2004, Awan et al., 2019, Heidemann et al., 2006 & Ovaliadis, Savage, and Kanakaris, 2010).

As a result, energy and resource capacities are seen as precious resources (Akyildiz, Pompili, and Melodia, 2004), and there is a need to understand how gains can be achieved in the lifetime of such SoS through identifying hardware and software design choices that can lead to improvements in energy efficiencies (Ovaliadis, Savage, and Kanakaris, 2010). For instance, a modelling method that leads to exploration of all the different ways to deploy software across the available hardware while achieving a certain level of resource utilisation and adhering to other non-functional requirements, becomes a significant tool to help the designer to make informed energy-efficient design choices. Furthermore, by enhancing such a modelling method to allow for exploration of SoS evolution delivers insight into whether those energy efficiencies can be maintained or, if not, where the drop off may start to occur along those evolution paths.

1.2 System Design Modelling

As part of modelling system designs there is a need to capture system characteristics and behaviours, both static in the architectural design and dynamic in the system execution, as part of providing the system designers with a complete view of the system performance and design choice impacts. Furthermore, the ability to start system testing earlier (i.e., at the point of conceptual design) and to combine models with production components provides a continuous test and build approach, and allows for early design risk identification.

Model Based Systems Engineering (MBSE) is a modelling approach fast gaining popularity to support system engineering efforts in dealing with system development sizes and complexities. It is largely centred around the move away from traditional documentation to the application of models to detail the numerous phases of the development life cycle. MBSE's main focus is on capturing static design details such as data modelling, requirements and architecture. The capturing of the dynamic aspects of the system design and use of model execution is an area of development (Friedenthal, Griego, and Sampson, 2009).

Pockets of work have seen MBSE approaches capture and investigate dynamic aspects of system design (Armstrong, 2008); however, the fact that abstract modelling is the foundation for these approaches reveals that there are difficulties in providing the system designers with appropriate insight into the lower level dynamic characteristics and behaviours of the COTS technologies found with modern SoS.

With the introduction of the Model-driven Engineering (MDE) philosophy to support the process and analysis of software system development, the use of such tools has led to the application of software performance prediction techniques (Kent, 2002; Barbierato, Gribaudo, and Iacono, 2011).

While MDE techniques are applied to the application of software development, being used to model and solve the configuration phases, as well as execution emulation, testing and analysis, its focus is purely on the software development life cycle. Overall system performance, which includes computing infrastructure, is not a consideration.

In contrast to the abstract approaches with the MBSE philosophy and the limited design focus on software systems alone with the MDE philosophy, System Execution Modelling (SEM) (Hill, Schmidt, and Slaby, 2009) is a development that moves from pure research into a measurement-based performance prediction technique designed to provide high-fidelity insight into overall system design performance.

As part of a broader research program looking into integrated SEM environments for SoS deployment and operational context performance insight, Falkner et al., 2013 introduces an architectural prototyping system capable of supporting scenario-driven experimentation for evaluating model suitability for deployment and real-time performance insight purposes. Falkner et al., 2014 also present an integrated approach to performance analysis and prediction of model-driven DRE defence systems centred on the use of the SEM paradigm. More recently, building on their previous research, Falkner et al., 2018 introduced a method for design

performance insight through an integrated model-driven approach for performance prediction of real-time embedded defence systems and SoS.

However, while these SEM approaches do introduce a new modelling and prediction techniques that considers the dynamic design aspects, overall system performance and operational contexts of SoS deployment, the approaches are predominantly focused on the representation of the system architecture and its workload. The SEM approaches do not consider the need to search through a large number of options associated with the deployment of large scale SoS, nor allow for methods whereby guides or bounds for software deployment can be based on known high-level requirements or constraints. Furthermore, the deployment techniques introduced with these SEM approaches are only manual in nature, leading to an inability to confidently search the large solution space for SoS deployment.

This thesis introduces an approach that addresses the identified capability gap for modelling deployment of large scale SoS within the SEM modelling paradigm introduced by Paunov et al., 2006, Falkner et al., 2014 and Falkner et al., 2018. The method described in this thesis will not only show the use of automation to search for SoS deployment options, but will also introduce an approach that will enable software deployment option exploration using high-level non-functional requirements and constraints modelling. The research outcomes from this thesis are also associated with the broader SoS SEM environments research program that has delivered key research outcomes by Falkner et al., 2013, Falkner et al., 2014 and Falkner et al., 2018.

This research will introduce a new modelling and prediction technique con-

sisting of two main contributions. The first research contribution introduces a new modelling approach that augments existing SEM modelling approaches for articulating system architectures and workloads. This consists of both pictorial and text-based modelling, and will allow for the definition of non-functional requirements and constraints associated with existing system architectures and workloads (through standard SEM modelling). The second research contribution introduces an automated learning mechanism to explore the large numbers of deployment options based on a system model, the run-time environment definition and modelled non-functional requirements and constraints. This new automated search capability injects itself seamlessly within the SEM modelling and prediction paradigm execution flow and is the key mechanism for developing the final SEM models ready for execution, rather than the extant manual approach via the SEM modelling environment.

As the new research detailed within this thesis is not introducing a new SEM approach, but rather looking to augment an existing SEM capability to close an identified gap, the research will build on an already-developed SEM environment. Furthermore, as this thesis is part of the broader research program that has delivered work from Paunov et al., 2006, Falkner et al., 2014 and Falkner et al., 2018, the SEM environment being built upon will be the MEDEA environment.

Using the proposed approaches allows system designers to guide and explore software deployment rapidly through automated search, based on high-level requirements and constraints, and gain insight into risks of performance impacts associated with those high-level design drivers and their associated low-level de-

sign choices. It also provides an ability to explore deployment options that would not commonly be found with manual approaches, where deployment is primarily guided through experience or previous deployment choices, thereby further reducing any risk of performance impacts from sub-optimal design choices.

Examples of non-functional requirements and constraints that could guide the software deployment options may include ensuring spare resource limits (possibly resulting from required power supply improvements or capacity degradation issues) are maintained for identified computing nodes. Overlaying these resource requirements are constraints for software component mandate deployment as a result of safety and system availability requirements. This results in the deployment solution search trying to satisfy both sets of requirements and constraints, or at least seek solutions that are optimised to support the greater number of requirements and constraints. Lastly, a safety critical thread is identified where a string of interfaces must adhere to temporal performances. Once again, the identified optimised deployment solutions seek to satisfy all, or as many as possible, of these constraints.

An extension to the optimal software deployment search capability is to add in the capability to have the search consider the evolution of the software system. In addition to considering high-level requirements and constraints, the optimisation search would also take into account system evolution characteristics and the resulting system evolution design options. In this scenario, potential design evolution paths would be explored, identified and deployed onto the existing run-time computing infrastructure for execution purposes. These evolved designs would

also include inherited high-level requirements and constraints (from the existing system design to be evolved) or potentially new ones. The executions would then be analysed to gain insight into the run-time computing infrastructure's ability to service the requirements of the evolved software system, as well as any impacts on the resulting overall system performance.

To enable the aforementioned system evolution consideration, this thesis introduces a new modelling method and exploration capability to identify possible new evolved software system designs, based on defined system evolution characteristics. The new method will once again augment an existing SEM modelling and prediction execution flow.

The new approach introduced with this thesis builds a new modelling language and set of modelling processes to enable definition of system design evolution characteristics. It also introduces a framework that allows the system evolution models to be interpreted to guide the search and creation of evolved system design options. Lastly, the entire capability is integrated within an existing SEM environment execution flow, which would also include the software deployment optimisation search capability, also introduced within this thesis. As a result, insight can be gained into how well existing computing infrastructure can house optimally-deployed evolved software system designs, while still satisfying high-level design drivers.

By delivering this new capability, system designers can gain insight into software system design evolution impacts on resource-constrained run-time environments. Furthermore, the approach promotes an ability to drive upgrade cycles of the underlying computing infrastructure (or parts thereof) based on the evolution

path and growth of the software system itself, and not based on a set period of time. In the case of design approaches based on decoupled periodic software and computing infrastructure upgrade cycles (Mitchell, 2010b; Mitchell, 2010a; Kerr, Phaal, and Probert, 2008; Boudreau, 2007), the time period for changing the underlying computing infrastructure could be extended leading to cost benefits and design change risk reductions. It would also provide insight into the life expectancy of non-recoverable host platforms (i.e., space vehicles), where the ability of fixed (non-up-gradable) computing infrastructure to host software system evolution options can be investigated and obsolescence forecasted.

1.3 Thesis Organisation

This thesis provides a survey of the literature highlighting the research gap, followed by the aim of this thesis in addressing that gap. A methodology chapter then provides a high-level introduction and reasoning about the methods used to develop the capability that will address the research gap.

A chapter on the new software deployment optimisation capability is then provided, followed by a chapter detailing the testing approach conducted and the results produced indicating correct behaviour and performance. This is further enhanced by a subsequent case study chapter and a demonstration of the new capability with an undersea network software system design and deployment scenario.

Following the software deployment optimisation chapters, a chapter is provided

introducing a new method and its use to explore system design evolution. This includes details on its application within a SEM environment, as well as development and integration details.

Finally, a conclusion and future works chapter is provided, along with technical detail annex chapters.

2. Literature Review

2.1 System Design Modelling

A modelling approach that captures system characteristics and behaviours, both static in architectural design and dynamic in system execution, provides system designers with a complete view of system performance and design choice impacts. Furthermore, it provides the ability to start system testing earlier (i.e., at the point of conceptual design) and when combined with production components provides a continuous test and build approach, which enables early design risk identification.

Model Based Systems Engineering (MBSE) is one such philosophy gaining popularity with system developers as part of trying to handle the growth in complexities in system designs. MBSE is an engineering approach that oversees the transition from traditional document-centric engineering methods to the formalised application of models to support the various system engineering processes starting at the conceptual design phase and continuing throughout the later development life cycle phases (Henderson and Salado, 2021).

Within the defence community there has been a wide spread adoption of MBSE

for such tasks as software and hardware configuration, product line modelling, requirements definition and data modelling (Hennig et al., 2016; Do, Cook, and Lay, 2014; Mitchell, 2011; Mitchell, 2010a; Saunders, 2003). While these methods provide well for capturing the static details of the system under design and their inter-relationships, the approaches lack in providing insight into the dynamic aspects of the system design.

MBSE approaches have been used to investigate dynamic aspects of system design (Perez, 2014; Do, Cook, and Lay, 2014; Armstrong, 2008). However, the fact that abstract modelling is used means large amounts of modelling effort and complexity are required to account for important low-level system details and dynamic behaviours. While the use of mathematical solvers can be incorporated to improve the fidelity of the simulations, its ability to account for lower level characteristics and behaviours of Commercial-Off-The-Shelf (COTS) technologies still lacks the required fidelity.

In the hope of introducing new MBSE capabilities for system developers based around Unified Modelling Language (UML), the Object Management Group (OMG) have introduced executable UML. Products such as the Cameo Simulation Toolkit[1] offer executable capabilities for system design investigation in areas such as verification, integration, temporal performance and deployment. Unfortunately, these capabilities are still abstract in nature and while additional mathematical solvers can be incorporated to improve the fidelity of the simulations, the ability to account for lower level characteristics and behaviours of COTS technologies still

[1] https://www.nomagic.com/product-addons/magicdraw-addons/cameo-simulation-toolkit

requires significant modelling efforts.

The prediction of software performance has developed from early approaches based on abstract models to model-driven engineering (MDE)-based approaches (Barbierato, Gribaudo, and Iacono, 2011). MDE is a software engineering philosophy that has a broader scope than just the architecture alone, as set out by the OMG's Model Driven Architecture (MDA) vision for direct use of models for software development and production. MDE considers both processes and analyses for software development. It is conducted around heavy tool use, and looks at elements such as modelling languages and models, and translations, as well as maintaining and coordinating construction and evolution for both models and languages (Akdur, Garousi, and Demirörs, 2018; Mohagheghi et al., 2013; Kent, 2002).

MDE techniques are typically applied to the development of application software components, but may also be used to model and solve the configuration and deployment phases, as well as system execution emulation, testing and analysis. This assists in the management of the entire software development life cycle.

System Execution Modelling (SEM) (Hill, Schmidt, and Slaby, 2009) is a development from research into measurement-based performance prediction that provides detailed early insight into the non-functional characteristics of DRE system design. SEM looks to utilise the benefit of abstract modelling where models are constructed relatively quickly at a fidelity level sufficient enough to answer system design questions, but executes those models within an actual run-time computing environment, such as those found with SoS constrained computing environments.

By doing this, SEM provides the required low-level insight on performance and behaviour of the COTS technology not seen with abstract modelling methods, while still providing the quick design change investigation capabilities seen with abstract modelling methods.

A SEM-based approach can support the evaluation of overall system performance, software integration, interactions and message exchange, and the performance impact of 3rd party software. At the core of this approach is the employment of simple models of resource consumption and behaviour extracted from the system's business logic. Using these models, a representative source code can be constructed and enable performance predictions analysis when deployed and executed upon real software and hardware test-beds.

SEM and MDE can be used in combination to support the emulation of system components and performance models, enabling performance data to be used to redesign and reconfigure the SoS, prior to any construction of the corresponding real system. The use of MDE, domain-specific modelling languages (DSMLs) (Paunov et al., 2006), automatic code generation, and the utilisation of COTS technologies have enabled the SEM approach to abstract the development complexities of DRE systems, while still ensuring detailed performance insight at the level required to provide performance evaluation of mission-critical system designs. However, work in this area is still in its early stages and more work is required to better support early performance prediction. In particular, there is a need to integrate the representation and visualisation of models and performance information to assist in early decision-making based on performance predictions using realistic

data sources (Edwards, Malek, and Medvidovic, 2007a), and to integrate SEM approaches and modelling environments with capabilities to support SoS considerations. Furthermore, work is needed to explore automated methods for deploying large-scale software system design models onto the available computing resources based on guidance from high-level requirements and constraints.

2.2 System Performance Prediction

As the software industry seeks to improve development processes and reduce costs, there has been a trend to transition from monolithic and proprietary type system designs to system designs that are modular, built from reused code or product lines and have a heavy reliance on standardisation (Kalske, Mäkitalo, and Mikkonen, 2018; Heydari, Mosleh, and Dalili, 2016; Balasubramanian et al., 2005). This shift in paradigm has also bought with it a critical need to understand performance or non-functional aspects of the design to minimise the risk of design change (Nawinna and Sandeepani, 2020; Lai et al., 2019; Martens, 2011; Balsamo et al., 2004), and a desire to do so as early as possible in the development cycle. As a result, the traditional software engineering methodologies that focus first on functional aspects of the system design and leave the non-functional aspects of the design to a 'fix-it-later' approach (Balsamo et al., 2004; Happe, Koziolek, and Reussner, 2011; Jasmine and Vasantha, 2007) are replaced with approaches that introduce performance evaluation early into the development cycle.

One of the earliest attempts to integrate performance understanding early within

the development cycle is the Software Performance Engineering (SPE) methodology (Smith, 1990). SPE allows for the modelling of software and hardware resource requirements and analysis of their performance through the use of two models: the software execution model and the system execution model. The first model details software execution behaviour with the use of execution graphs, while the use of Queuing Networks enables the system execution model to represent the platform under analysis, including both software and hardware.

Smith and Williams, 1997 developed an approach for performance evaluation of object-oriented systems using the SPE-ED tool. Using automation to generate queuing networks from user developed behaviour scenarios or execution graphs, it provides insight into end-to-end response times, elapse time for each processing step, device utilisation and time spent at each computing device. However, this approach needs expertise in low-level knowledge on resource utilisation to derive the resource requirement model details for each software resource request.

An improvement on the SPE-ED approach was made with the direct use of UML diagrams for the development of the two SPE models. Cortellessa and Mirandola, 2002 developed the methodology known as PIMA-UML (Performance IncreMental vAlidation in UML) to utilise standard UML notation diagrams. Using enhanced UML diagrams to generate extended queuing network performance models and classical computing performance analysis techniques (Nance, 1990; Lavenberg, 1983), software engineers are provided with early and incremental performance evaluations throughout the development cycle. This approach was further evolved to consider mobile software architecture performance insights

(Grassi and Mirandola, 2002). Once again, as was the case for SPE-ED, PRIMA-UML requires expert knowledge to ensure models are detailed correctly to realise accurate performance indicators.

There have been various UML-based approaches for performance modelling that have recognised the need for standards-based notations to enable modelling to occur directly from standards-based system design artefacts. This has been exemplified with the Object Management Group (OMG) introduction of UML profiles that focus on non-functional descriptions of system designs. Li, Wang, and Shi, 2013; Tabuchi, Sato, and Nakamura, 2005; Distefano et al., 2004 introduced the use of the now obsolete UML SPT (Schedulability, Performance and Time) (OMG, 2003) profile to enable the modelling of performance early within the development cycle. Distefano et al., 2004 detailed the first step of an automated performance measurement process that contains a mapping of the UML SPT profile to the Performance Context Model (PCM). While the use of an intermediate model translates well for cross performance modelling platform capabilities, the use of the obsolete SPT profile means that design artefacts based on newer UML profiles would incur a further translation and the potential occurrence of misinterpretation or information lost with profile mapping.

In 2008 the OMG introduced the UML MARTE (OMG, 2011) profile to replace the SPT profile and delivered improved capabilities in UML in the domain of model-driven development of real-time and embedded systems (RTES). The aim of the new profile was to provide a mechanism to address analysis information requirements through model annotation. Liehr and Buchenrieder, 2009; Gilmore et al., 2011 and

Chise and Jurca, 2009 used the MARTE profile to develop UML descriptions of the systems under development and then generate performance models (such as extended queuing network model, analytical models or simulations) subsequently to provide early performance insight.

Herrera et al., 2014 presents an approach for design capture and automated performance model construction in support of exploring design options and complexities in addressing tight performance requirements of embedded system design. While Shailesh, Nayak, and Prasad, 2020 presents research based on the MARTE profile annotated with UML sequence diagrams to enable automated evaluation of system performance. Through model transformation methods, the meta-model-based approach allows for mapping of sequence diagram information to Petri net performance models, and performance insights, such as utilisation and throughput.

Ding, 2016 also introduces the use of Petri nets based modelling and analysis to investigate dynamic aspects of software architectures associated cloud environments.

While these methodologies deliver the required early predictive insight into performance, the majority of system level performance predictions do not consider the complex characteristics of component-based systems and their interactions with the operational computing environment (Kuperberg, Krogmann, and Reussner, 2008; Hill and Gokhale, 2008). In addition, while some considered the operational environments characteristics from a high level, such as the SPE system execution model, they did not translate well for accounting for the various configurations of the operational computing environments. There is also little ability within the

modelling frameworks to account for the complex nature with which a component interacts with its operational computing environment and the number of deployment permutations a component-based system could have, as well as the impact on the overall performance of the system (Kuperberg, Krogmann, and Reussner, 2008; Koziolek, 2010).

An alternative methodology to UML-based notation model development is the use of existing or development of new architecture description languages (ADL). These extant ADL's or newly created ADL's, when integrated with performance modelling tool suites, also support performance analysis requirements.

The use of the Architecture Analysis and Design Language (AADL) models (SAE, 2007; Feiler, Gluch, and Hudak, 2006) with the Ocarina tool suite allowed Hugues et al., 2007 to develop a rapid prototyping process for the development of DRE systems. Focusing on the design-by-refinement approach, this methodology saw the development of AADL models and the assessment of those models for semantic correctness and schedulability analysis for execution deadline times.

Hemer and Ding, 2009 developed an ADL called CRADLE whose aim was to support improved architecture analysis, adaptation, and expressiveness of systems. The ADL enabled models and architectures to be detailed in a more generic fashion, thereby paving a way for a single model to instantiate the various compositions of system architecture via parameterisation. The approach provides effective insight into the interactions within the system architecture from the viewpoint of component numbers, connectors, connection topology and communication protocols, but does not account well for other non-functional aspects of the system

design.

The introduction of domain specific modelling languages (DSML) and their ability to better define problem domains or tasking (as compared with general-purpose programming languages) in conjunction with the increasing use of model-driven engineering (MDE) and new MDE technologies, led to the creation of DSML development environments (Amyot, Farah, and Roy, 2006). These DSML development environments therefore provided the capability to map, extract and enhance existing ADL's in way that catered for closer descriptive alignment of the problem domain without having to create a totally new ADL.

Edwards, Malek, and Medvidovic, 2007b avoided the requirement to design an architecture language from scratch by developing XTEAM (eXtensible Tool-chain for Evaluation of Architectural Models) through the use of DSML mapping. The approach provides a way to address detail shortfalls in existing ADLs, as well as the inflexible notations and narrow vocabulary, while making the use of MDE to develop XTEAM models and generates executable simulation models for dynamic analysis. Using the Generic Modelling Environment (GME) (Vanderbilt, 2008), the authors were able to map ADL's to create the XTEAM ADL, while the XTEAM model interpreter framework enabled the transformation of the XTEAM models to executable analysis models. The approach provides effective insight into the non-functional design aspects of energy consumption, reliability, latency and memory usage, but does not consider the performance characteristics associated with different elements of the operational computing environment, such as middleware.

Heinrich et al., 2018 introduces a tool for software architecture quality inves-

tigation based on the Palladio Component Model (PCM) (Reussner et al., 2011). Using the PCM as a domain-specific language, various analysis techniques (e.g., Petri Nets, Queuing Network) are conducted through model transformation from context views of the system and operational context. These provide insights on system performance and how they trace to certain individual system components of the system under investigation.

While the introduction of ADLs and DSML provides closer definitions of concepts and relationships associated with non-functional aspects of the system design, there is still little consideration for the complex nature of component interactions within the system infrastructure. Furthermore, abstracted analysis methodologies like Petri Net analysis or the use of simulation and other high-level formal analysis approaches do not translate well to enable analysis of lower level elements of the operational computing environment. The approaches fail to account for the way components are deployed within their run-time environment and the fact that the run-time environment could be heterogeneous in nature with varying computing platform performances.

In an effort to address the need for performance analysis focusing on the lower level elements of the operational environment, such as middleware, the approach of utilising benchmark testing and embedding in performance models was established.

Liu et al. (Liu, Fekete, and Gorton, 2004) demonstrate such a technique in response to the industry trend of incorporating increased use of middleware architectures as a cost reduction strategy for distributed applications. Liu, Fekete, and Gorton, 2004 demonstrate the use of queuing networks and state-machines

to detail the system architecture and behaviour of service components and then characterised the application's infrastructure or service components' utilisation. The three modelling and characterisation steps then lead to the derivation of the performance equation. With the use of middleware bench-marking within a controlled test environment, the performance equation was populated with the performance parameters needed to conduct the performance analysis.

Evangelista, 2021 introduce a method for performance modelling associated with database management and the challenge of correct configuration for NoSQL implementations. Through the use of Petri Nets and Queuing Networks models in this instance, insights into cloud-based NoSQL deployment are delivered and shown to provide benefit in addressing the challenges of preventing bad behaviour and costs impacts to the users of such services. While Iakushkin, Shichkina, and Sedova, 2016; Faisal, Petriu, and Woodside, 2014 also make use of Petri Nets and Queuing Networks to enable performance modelling of middleware classes and implementations, as part of component-level interaction investigation and design decision support to consider performance aspects like utilisation, scaling and QoS.

Once again, these approaches were limited in accountability for lower level details of the run-time environment, such as the cross platform and dynamic nature of a software component-based run-time environment. While the modelling approaches provide high fidelity analysis of the software infrastructure performance, the approaches do not translate well to heterogeneous environments. The process of integrating the bench-marked performance metrics into performance equations leads to higher quality results, but the requirement to conduct bench-marking for

each type of middleware and execution scenarios does not translate well to heterogeneous environments. The time consuming nature of setting up and configuring controlled test environments and test applications impacts the quick turnaround benefit of abstract performance modelling and analysis.

2.3 Component-Based System Performance Prediction

A survey conducted by Koziolek (Koziolek, 2010) observed that the majority of the proposed research for the evaluation of non-functional aspects of component-based systems could be categorised into either performance prediction or performance measurement. These could be further broken down into approaches based on UML or a proprietary meta-model, and focused on middleware, formal specification or measurements of system implementation.

As detailed in Section 2.1, a number of approaches based on UML allow for the abstraction of details and analysis, thereby providing reasonably quick turnaround time for insight into performance of the design. However, they fail to capture the details of the characteristics of how components can be deployed. In response, Kaur, 2012; Jasmine and Vasantha, 2007; Balsamo, Marzolla, and Mirandola, 2006; Bertolino and Mirandola, 2003 introduced UML approaches that succeeded in capturing both the required component-based information within the models and the subsequent analysis. While succeeding in capturing the relevant information, an open problem remains with the ability of these approaches to account for the

dynamic and 3rd party nature of the operational environment and its interaction with the system components (Becker et al., 2006).

Furthermore, as highlighted by Ozkaya, 2018, UML-based languages have been adopted to domains like embedded and real-time systems, but a limited number provide behavioural modelling viewpoints for design aspects such as deployment and operation. The survey also shows there is limited support for simulation and ability to gain insight into system dynamic aspects.

A complementary approach to establish modelling notations and analysis techniques is introduced by Grassi et al. (Grassi et al., 2008; Grassi, Mirandola, and Sabetta, 2005; Grassi, Mirandola, and Sabetta, 2007) who proposes a kernel language known as KLAPER (Kernel LAnguage for PEformance and Reliability analysis). KLAPER is an intermediate language that allows for the mapping of design-oriented notations to analysis-oriented notations in a decoupled manner. Ciancone et al., 2014; Ciancone et al., 2011 also introduces a KLAPER language tool-suite to support reliability and QoS investigations for component-based system designs and assemblies.

The use of a Meta Object Facility (MOF) meta-model-based approach enables bridging of various design-model methodologies, with their many design-level notations, to various analysis methodologies. It also paves the way for reductions in effort and complexity when exploiting MOF-based model transformations frameworks. Furthermore, the kernel language approach provides a way to capture the relevant aspects and information of component characteristics for analysis, while discarding unnecessary information.

While the kernel language addresses the need to define specific component oriented details and related analysis, the approach is limited in regards to detailing the dynamic behaviour of components. Since the focus is on the mapping of extant design-level notations and analysis methodologies, the same observations made about the use of abstract approaches and their inability to account for the run-time environment remains.

Performance prediction approaches with a middleware focus have gone part way to delivering insight into the deployment environment and its impact on component performance. The process of identifying performance specific middleware characteristics and feeding them into analysis models allows for insight into performance aspects of the lower level infrastructure elements (Abdelaziz, Kadir, and Osman, 2011). This then provides insight into the impact on component behaviour and overall performance of a particular design. Chen et al., 2005 extends the work of Liu, Fekete, and Gorton, 2004 to consider the unique characteristics of component-middleware. While Rathfelder et al., 2014 introduces work that looks to model and predict performances for component-based systems, and the influence of publish-subscribe middleware characteristics (through developed information repositories) has on communications.

Eismann et al., 2018 introduces work that allows for component-level dependencies to be modelled, model transformation, and Queuing Petri Net-based analysis to explore the interactions between components and influence on system performance, such as bottlenecks and utilisation.

Zhang et al., 2006 develops an approach that demonstrated the use of built-

in architectural pattern libraries to capture the structure and behaviour of the middleware of interest. It also provides a mechanism to have this information automatically integrated into the UML-based application model for analysis.

Each approach tackles the problem of capturing the underlying computing infrastructure performance by either capturing characteristics through measurement, or using architectural patterns. However, they lead to a heavy-handed approach to the analysis of heterogeneous operational environments. Furthermore, the need for configuration and measurement, or use of static structure and behaviour information, does not translate well for the dynamics involved with component redeployment or changes in the hardware platforms. Each approach fails to capture details and provide insight into the deployment and the redeployment of components across more than one middleware and hardware platform within a single operational scenario, as well as the resultant performance impact.

When trying to understand the performance of component-based systems, current performance prediction techniques need to firstly recognise that the system behaviour characteristics are predominantly a result of the manner in which the components (which are assembled to satisfy the application's functional requirements) are deployed and interact with the operational environment (Balsamo et al., 2004; Kuperberg, Krogmann, and Reussner, 2008; Chen et al., 2002). Secondly, and probably the most important, the prediction approaches have to consider the emerging trend of reliance on third-party infrastructure software within component-based heterogeneous systems; meaning the ability to provide accurate performance details for embedding into abstracted models becomes difficult, as a direct result of

there being little access to third-party performance details. It therefore becomes evident that the only way to definitely understand the non-functional requirements of such systems is through measurement-based approaches (Becker et al., 2006).

If the above observation is the driving force for shaping a performance prediction technique for understanding stringent mission-critical system performance requirements, limitations still exist with measurement techniques such as those introduced by Rathfelder et al., 2014; Zhang et al., 2006; Chen et al., 2005; Liu, Fekete, and Gorton, 2004. These approaches involve a heavy-handed measurement and analysis process for gaining insight into performances of heterogeneous platforms and use for overall system performance prediction. Therefore, the measurement-based approach will have to be a mixed approach of measurement-based with computing environment prototyping and modelling to allow for insight into the complexities of the deployment characteristics, while still ensuring the workload and design change analysis can occur easily. This then leads to the ability to provide relatively high fidelity system performance prediction inputs into the design phase of the development cycle (Koziolek, 2010).

2.4 System Execution Modelling (SEM)

SEM is an approach that proposes to exploit the construction of workload models based on the internal logic of the software components to be analysed (Denaro, Polini, and Emmerich, 2004). It allows system matter experts to model the software system accurately before it is developed, and to capture software, hardware, and

middleware constraints and perspectives that are impossible to capture in analytical models. Denaro, Polini, and Emmerich, 2004, Paunov et al., 2006 and Hill et al., 2010 define initial approaches for system execution modelling, employing the use of prototyping to gain a detailed understanding of system design performance. Denaro, Polini, and Emmerich, 2004 construct component-level temporal performance models outside of the internal business logic, while Denaro, Polini, and Emmerich, 2004 exploit the construction of workload models based on internal business logic to provide more detailed and complete insight into the overall system performance.

Slaby et al., 2006 introduce a measurement-based approach that allows software and system engineers to conduct quantitative performance analysis, obtaining crucial insight into performance issues associated with a system design, early in the development cycle. The primary focus of this research is enterprise DRE systems. The component workload emulator (CoWorkEr) utilisation suite (CUTS) (Hill and Gokhale, 2008) is an implementation of this approach. The CUTS environment employs predefined workloads for software components from which code is generated, and integrates automated tools for the building, configuration, and deployment of generated code on various hardware, software, and middleware infrastructures. The system execution modelling tool suite significantly reduces experimentation and analysis efforts early within the development cycle of a system of interest. However, it only employs simplistic workloads focused only on CPU and memory consumption alone. For example, a CPU-focused workload for a software component would be defined and implemented as a raw busy-waiting

time, e.g. wait = 5ms. This in turn limits its performance analysis capabilities, with only simple aggregates of utilisation and network bandwidth consumption available as performance metrics.

As part of a broader research program looking into integrated SEM environments for SoS deployment and operational context performance insight research from Falkner et al., 2013, Falkner et al., 2014 and Falkner et al., 2018 expanded the focus of the SEM research from Slaby et al., 2006. The aim of this work, and the overall research program, introduced approaches that allow for analysis at the SoS level, and support for scenario-driven experimentation for evaluating model suitability for deployment and real-time performance insight purposes. The research program presented a model-driven performance prediction system that utilised realistic data sources and provide improved methods for visualisation of the causes of performance issues, as well as relationships between the models and performance constraints. The research program enabled performance prediction to occur under a variety of conditions, using data obtained from real, emulated, or simulated sources. The research program also introduced the use of Domain Specific Modelling Languages (DSML) for elements such as scenario definition, performance requirement and metric definitions, as well as system design evolution investigation.

In addition to contributing to the overall research program and research outcomes delivered by Falkner et al., 2013, Falkner et al., 2014 and Falkner et al., 2018, this thesis introduces research outcomes that delivers the identified system design evolution investigations capabilities within the SEM paradigm. Based around a

system design evolution DSML, model translation, interpretation and creation, this body of research is presented in Chapter 8.

While Falkner et al., 2013, Falkner et al., 2014 and Falkner et al., 2018 introduced methods for model-driven large scale system design performance prediction, that utilises real or synthetic data sources, the complexity of dealing with large scale deployment of software is still a largely a manual process. When dealing with small scale models, exploring all the possible software deployment options with a manual process is viable. As the modelling scale increases and the number of options for software deployment grows significantly, manual methods are less viable and the questions about the ability to identify appropriate deployment options from the large solution space appears. Therefore, this presents the need to have an ability to explore the benefit of search algorithms and automation for identifying software deployment.

The ability to combine the developed SEM modelling approaches (Falkner et al., 2013; Falkner et al., 2014; Falkner et al., 2018) with an automated optimisation or learning mechanism would reduce the burden for designers investigating large scale system designs. It would also allow for improved confidence in investigation outcomes through the ability to search significantly larger sets of solutions. The capability would also pave the way to develop methods that would allow for software deployment options to be explored and identified based on high level guidance for certain goals. This would provide particular benefit to system designers, by allowing them to drive automated searches with high-level design requirements and constraints, and to understand the impact of different design choices. An

example of this could be understanding how many different deployments exist that would satisfy system end-to-end latency while ensuring spare resource limits are maintained, or needing to adhere to mandated deployment constraints.

Furthermore, benefit can be provided to system designers when considering the upgrade cycle aspects of the system and how deployment might occur as the system evolves. By providing a DSML to detail potential evolution aspects for the system design, this model could then be tied in with the deployment optimisation capability, which, when integrated with a SEM modelling approach, would provide insight into how a constrained DRE SoS would provide capability throughout its lifetime.

2.5 Software Deployment Optimisation

Optimisation is a process whereby actions are taken to make the best or most effective use of a situation or resource for a known function and desired outputs, the task is to find the actions or inputs leading to those outputs (Eiben and Smith, 2003). For a system this maybe to make modifications to achieve its best design options in response to a set of criteria. In the case of software deployment optimisation this could be to modify and search through the different ways the software can be deployed within its computing environment with the aim of minimising or maximising certain system features that are associated with a set of identified system requirements or constraints.

As the complexity of system designs continue to increase, the complexity of

search processes to identify best or most effective design outcomes also increases. As a result, the use of search techniques like evolutionary algorithm searching is emerging and gaining popularity for exploring solutions for real world complex problems (Vikhar, 2016).

Evolutionary algorithms, which are a subset of evolutionary computation, conduct solution searches through an evolution mechanism inline with what can be observed through biological evolution. Through a process of reproduction, mutation, re-combination and selection, the solutions produced within each search cycle should evolve to be better than the previous set of solutions. In general, an evolutionary algorithm implementation is a global search algorithm where an initial population (set of solutions) is randomly generated. This is the first generation. Subsequent generations are then produced through an iterative process (up to the required number of generations) of solution fitness evaluation (a score to indicate how good the solution is), selection of the fittest solutions (parents) for reproduction, generation of children through crossover and mutation of selected parent solutions, and finally replacement of least fit solutions with newly created more fit solutions. Genetic algorithms are one of the most commonly used approaches for optimisation and search problems (De Jong, 2006; Eiben and Smith, 2003; Vikhar, 2016).

In addition to the global optimisation search approaches, where solutions within a generation are combined and mutated in an effort to gain better solutions, local optimisation searches can also be used. A local optimisation search considers only the individual solution within a generation, where crossover and mutation can occur for its solution only to see if a more fit solution can be produced. If the

new solution fitness is better than the original individual solution it replaces that original solution (Hoos and Sttzle, 2004).

Evolutionary algorithms have been used for deployment optimisation purposes to identify better ways to deploy wireless networks for such things as node numbers, throughput, cost or transmission delay (Wang et al., 2017; Rebai et al., 2015; Budi et al., 2014). Similarly, the use of evolutionary algorithms have been designed to consider better ways to deploy cloud environments. Chen et al., 2012; Liu, Lu, and Liu, 2010; Li et al., 2009 introduces the use of evolutionary algorithms to reduce the costs by identifying where and when to place cloud computing infrastructure (i.e. virtual machines and services), while Zhu et al., 2013 introduced a cost reduction method utilising a genetic algorithm for optimising deployment of cloud services across distributed data centres. Vanrompay, Rigole, and Berbers, 2008 also made use of evolutionary algorithms to identify deployment profiles for services onto a set of connected nodes to allow for a certain level of Quality of Service (QoS) to exist while still minimising the communication cost between the chosen nodes. While these approaches do consider the influence of non-functional requirements (such as temporal performance), the level of these optimisations searches is at the system or platform granularity and thus they do not consider the lower level elements that make up those systems or platforms.

Peng et al., 2010 introduced deployment optimisation for mapping software components to automobile electronic control units (ECU) built around the AUTOSAR (Automotive Open System Architecture) project. The research applies an evolutionary algorithm to search for solutions to deployment profiles that minimise

the communication costs between ECU nodes, while also considering resource requirements and scheduling. The approach considers resource demands at the individual component level, as well as the communications between components, which then aggregate to produce the whole ECU node communication utilisation. However, this approach does not consider temporal performance non-functional requirements, such as end-to-end temporal performance of message exchanges across a chain of components.

Meedeniya et al., 2011 introduces an approach that utilises the ArcheOpterix framework[2] that also looks into deployment of software components within a ECU environments. The focus of this research is to optimise their deployment around service reliability requirements. Using specifications from an AADL system design specifications, software and hardware architecture details, along with reliability-based attributes are extracted. These are then used to establish an objective function for use in combination with a genetic algorithm to identify deployment options based around service reliability. The approach does consider behaviour and performance down to the software component level, but also suffers from not considering interactions and end-to-end performances across multiple components strung together.

Zimmer et al., 2012 also looks at evolutionary algorithms for optimising deployment of components to an ECU. Once again geared around the AUTOSAR project, the approach taken with this research is based around the ability to satisfy safety requirements. This approach too considers the communication between

[2]Available on the web http:mercury.it.swin.edu.aug archeopterix

components deployed, as well as constraints and resource demand, but once again does not look into temporal performance requirements such as end-to-end timings.

White et al., 2011 presents research looking into the use of a hybrid heuristics/evolutionary algorithm to evaluate optimising deployment within a DRE space system with respect to the energy consumption footprint for the topology. Known as ScatterD, the search looks at attributes and behaviours at the software component level, such as tasks and CPU consumption, message rates, size, type and message interactions between components. It also explores spatial deployment plans that minimize power consumption while also considering network communication requirements in combination with resources requirements and constraints. However, the network communication performance only considers interactions between software component pairs and does not look at performance across a critical string of components. This is also observed with the work introduced by Nazari et al., 2016. In this work the definition of constraints and performance requirements that lead to deployment options, via the use of a mixed integer linear programming model, are focused on the interactions between components, and not across a series of components of interest.

2.6 System Design Evolution Modelling

In support of the broader research program looking into integrated SEM environments for SoS deployment and operational context performance, this thesis introduces a new method that allows for definition of software system evolution

that would feed into a SEM environment execution flow. To this end, this thesis explored the literature to determine existing methods for software system evolution prediction and modelling. This included looking at modelling contexts, system evolution focus areas, levels of model fidelity and the ability to predict performance for the entire system evolution.

Research into methods for exploring system evolution can largely be broken into two main approaches: the forensic approach that looks back at system evolution and observed characteristics, or the forward looking approach of predicting how systems evolution could occur.

Baxter et al., 2006 conduct an in-depth study into the structure of Java programs deployed over time. Through detailing structural characteristics of Java programs and application-related variations, the research provides a basis as to how and why certain structures occurs, as well as their management. Li et al., 2006; Lavery et al., 2004 adopt a similar type of investigation by looking at design changes with the introduction of factors such as legacy systems, security, concurrency and open-sourcing, as well as the response to system bugs. Godfrey and Tu, 2001 look at the case study of the Linux kernel evolution. Using a tool known as Beagle, insight into how such a large system changed over time is provided.

While these approaches provide insight into common evolution paths of systems and impacts, the investigations are conducted post system evolution and are unable to provide support to system designers on system evolution options during the system's upgrade life cycle phase.

Selic, 1998 introduces a UML-based approach that facilitates software archi-

tecture definition for initial design and evolution. By detailing system component relationships and behaviours with abstract modelling constructs, the approach constructs and executes models to give insight into possible design changes and evolutions. Medvidovic, Rosenblum, and Taylor, 1999 extend this approach by introducing ADL-based descriptions within a Java environment to enable model analysis, evolution and implementations of possible software architectures. However, these approaches are built on abstract modelling methods and are limited in their ability to account for the dynamic low-level performances and behaviours of the run-time computing infrastructure.

Franco, Barbosa, and Zenha-Rela, 2013 introduces an ADL-based approach to investigate reliability and other system quality attributes as a result of architectural-level evolution. The work presents a method for ADL annotations to consider control flow of inputs and outputs, usage profiles and component reliability specifications. These definitions then translate for use with a stochastic model (via the PRISM tool (Kwiatkowska, Norman, and Parker, 2009)) to predict the impact of component reliability, usage profile and system structure on the overall reliability. As a result, insights are gain into performance shortfalls and support decisions on how best to evolve a system.

Zhao et al., 2007 introduces an approach to allow for design pattern-based investigation into system evolution via graphical design methods. Motivated by identified shortfalls of UML-based descriptions, they show how defined evolution paths for a set of design patterns, and transformation rules evolve designs automatically through graph transformations mechanisms. While Ulbrich et al., 2010

introduces work on using the MARTE profile for modelling existing real-time system code base and it evolution, via auto-generation mechanisms, without creating and analysing new evolved code baselines.

Koziolek et al., 2011 introduces a predictive approach through their integrated framework for performance and reliability prediction of evolving system architectures. Using model transformation, the approach creates Palladio Component Models (PCM) (Becker, Koziolek, and Reussner, 2007) from an architecture meta model known as the Service Architecture Meta Model (SAMM) (Becker et al., 2008). The SAMMPCMs form the evolution scenarios, where performance is predicted with abstract model solvers. Koziolek et al., 2013 builds on this research further with the application of SAMMPCMs for a control system evolution performance prediction case study.

However these approaches still lead to a limited ability to account for the low-level dynamic aspects of an evolved system design and its non-functional performance.

2.7 Research gap

The literature shows that there are a small number of pockets of research on the use of measurement-based techniques for performance prediction of a mission-critical DRE system design early in the development cycle. Furthermore, the literature shows that system execution modelling techniques lead the way in accounting for low-level behaviours and performances when wanting to evaluate the relationships

between business logic and resource demands, along with other non-functional requirements.

While the outcomes from system execution modelling research are better suited to provide the type and level of design change impact insight required for large, complex, DRE system designs, there is a capability gap in being able to manage the complexity and scale of large systems. Further improvements were made with research from Falkner et al., 2013, Falkner et al., 2014 and Falkner et al., 2018 as part of a broader research program looking into integrated SEM environments for SoS deployment and operational context performance. However, the ability to handle complexities associated with deployment of large scale SoS requires further research in order to introduce targeted and automated deployment of software across an available run-time computing environment.

The literature shows that the application of deployment optimisation based around non-functional requirements and constraints is evident in many computing paradigms. However, large areas of research consider a granularity for deployment optimisation at the system or platform level. For those cases where software component granularity is present, the deployment optimisation is based around the individual elements largely in isolation, rather than through broader system aspects, such as end-to-end temporal performance.

It is also evident from the literature that if system evolution performance prediction is required, of the methods that are forward looking rather than a postmortem, research in these areas tends to be largely abstract in nature. To maintain the level of fidelity required for the performance insight associated with

mission-critical DRE SoS design evolution, an evolution prediction approach would be required to for integration into a SEM type environment.

3. Aim

The opening two chapters of this thesis detailed how SoS find themselves in a continual state of change in response to evolving user requirements or operational environment condition changes. Furthermore, when dealing with constrained SoS, where there is no ability to add to its computing processing capability, non-functional aspects of the architecture become a major design concern for the initial design and subsequent evolution.

When employing modelling methods to gain insight into these non-functional design aspects, it was also identified that there is a need to use a modelling method that enables identification of design risks early in the development and integration cycles for both the initial development and any subsequent upgrades. In addition, by allowing for mapping of models onto real computing platforms for evaluation of non-functional requirements and constraint satisfaction, significant benefit is gained from earlier identification of design risks associated with constrained DRE systems or SoS through measurement-based analysis.

Based on the literature search, pockets of performance prediction research exist that focus on measurement-based techniques to deliver the level of fidelity

required for early insight into systems and SoS design risks. Furthermore, the measurement-based SEM technique led the way in providing insight into the low-level behaviours and performances related to key non-functional requirements associated with constrained SoS. However, this technique has a gap in capability when dealing with the complexity of large scale SoS designs and how best to deploy the large number of software components across a constrained computing environment.

Furthermore, the literature indicates that system evolution performance prediction research is largely abstract and not to the fidelity required for performance insight for mission-critical SoS design. As a result, introducing a method for system design evolution prediction with a SEM paradigm modelling environment would enable high-fidelity mission-critical SoS design evolution performance prediction and insight.

In response, this thesis introduces research that will deliver an automated approach for optimised performance prediction of evolving DRE systems or SoS designs. It will detail a method that utilises domain-specific modelling languages, model-driven engineering, and deployment optimisation. The approach will be a measurement-based prediction approach that allows for a 'what-if' analysis of optimal deployment of software guided from non-functional requirements and constraints associated with an evolved system or SoS design.

As a result, system designers can initially explore how best to deploy software on available computing resources, while satisfying high-level requirements and constraints. Then, in contrast with periodic-based upgrade cycles for computing

resources (or the host platform when dealing with fixed constrained computing environments), upgrade cycles can occur based on software system evolution paths and an ability for the current computing resources to still service high-level requirements and constraints.

An example of this would be looking at how a software system could be optimally deployed onto a constrained computing environment, followed by wanting to understand how changing software resource footprints become an issue and impact overall system performance. By allowing for modelling of evolution trends associated with software system design, investigations can occur as to the possible ways in which the software system design may evolve and what performances would be expected when deployed. It would then allow for prediction of life expectancy of fixed computing infrastructure.

Furthermore, in the case of fixed or constrained computing environment this insight would also lead to exploration of the life expectancy of the operationally deployed host platform. It could be used to investigate methods for load shedding, possibly removing functionality, or, potentially, introduction of additional host platforms and new computing capabilities under a distributed system paradigm.

The approach will firstly look to develop an evolutionary algorithm optimisation technique for software components to be deployed onto available run-time computing nodes, based on high-level non-functional requirements and constraints. Examples of non-functional requirements and constraints would be temporal performance, SWaP budgets and mandated deployment needs.

This thesis will also introduce a new instrumentation framework capability.

In the first instance this capability will allow for confirmation that the objective function and search algorithm are behaving and performing as expected. It will then provide insight into the capability of the search and performance of the optimal software deployment solutions, relative to the defined non-functional requirements and constraints.

The second part of this thesis research evaluates the development a new DSML that will allow system designers to explore system evolution for particular parts of the system or SoS design. The approach will be based around information extraction from an extant design model, augmentation with the new DSML, and creation of a new model to describe areas and characteristics of the possible system evolution. As a result, this will guide exploration into options for how the system design may evolve. These evolved designs can then be deployed optimally and deliver insight into their resultant performances. The DSML will allow for definition of architectural changes, behaviour shifts and evolving resource constraints. This would include decomposition of software components, improving failure resilience, possible resource footprint growth, temporal performance improvements, evolving SWaP budgets and evolving mandate deployment needs.

4. Methodology

4.1 Software Deployment Optimisation Methodology

Figure 4.1 shows a generic high-level execution flow for the SEM method. As can be seen, this execution flow is broken up into two main phases. The first phase is the system under investigation definition through modelling, where the modelling is completely manual and flows through four different system design context views. The second phase is the construction of executable code, based on the developed model, followed by the deployment of all the code artefacts in support of execution on the available computing environment. This phase also includes the generation and storage of all instrumentation data taken during the execution. Unlike the first phase, this phase is completely automated.

In response to the capability gap identified previously, this thesis is introducing an automated method to replace the existing manual and unbounded approach of defining how software is deployed onto the available computing environment for

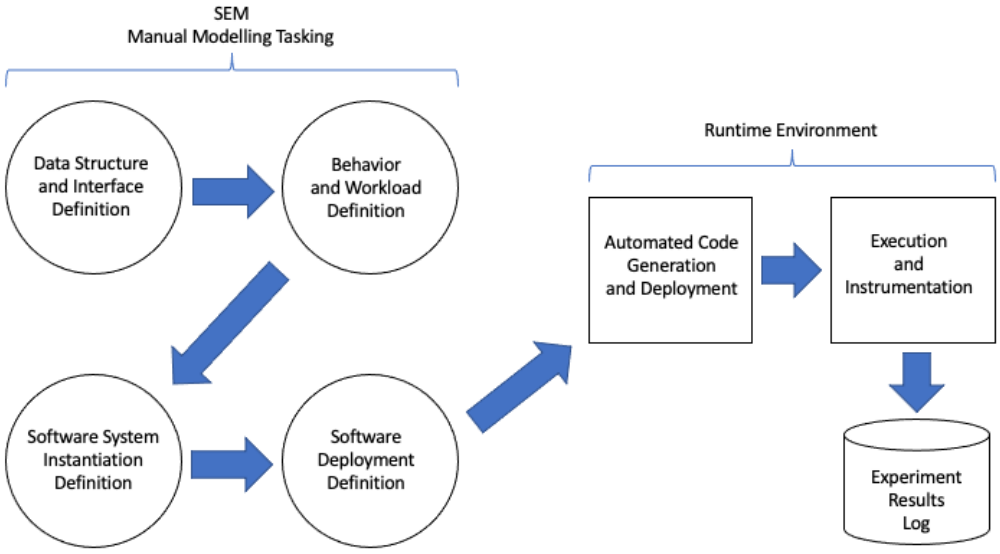

Figure 4.1: Generic high-level SEM execution flow behaviour

the system under investigation. Furthermore, this new automated method will use high-level requirements and constraints to guide the identification of deployment options and resultant performance.

To execute this new capability the introduced approach removes the need for conducting any manual software deployment modelling that normally occurs with current SEM modelling practices. It then augments the modelling processes within the Software System Instantiating Definition context view with new modelling elements and processes to address the information gap associated with investigating new system performance areas. A method is then introduced to extract pertinent details from the overall system model. This extraction then enables the creation of new interim models capable of holding the information needed for an automated search method further down the execution flow path.

Following the addition of new methods for the modelling phase, this thesis introduces the use of a machine-based solution search technique. By applying an automated search technique, the capability is able to work through the large solution space for software deployment options and identify those solution that perform best while satisfying high-level requirements and constraints. Based on the use of evolutionary computation along with a developed scoring algorithm, the search will be able to determine the system performances and how well a software deployment profile satisfies the modelled high-level requirements and constraints.

As a result, the modelling and analysis effort for large software system or SoS deployment is simplified by removing the complexities and effort associated with achieving this manually.

With the solutions identified, the new method then utilises model interpretation and construction to enable integration with the desired SEM environment. This results in the construction of SEM models with the optimised software deployment profiles for the available resources found within the detailed computing environment (i.e., from the original SEM model).

The last part of this new software deployment optimised capability is the introduction of new instrumentation techniques to enable evaluation of the new system performance areas associated with the high-level requirements and constraints. Once again, this new method augments the existing instrumentation framework found within the identified SEM environment.

As indicated previously, the research from this thesis is part of a broader SEM research program. Another research outcome of this program was the development

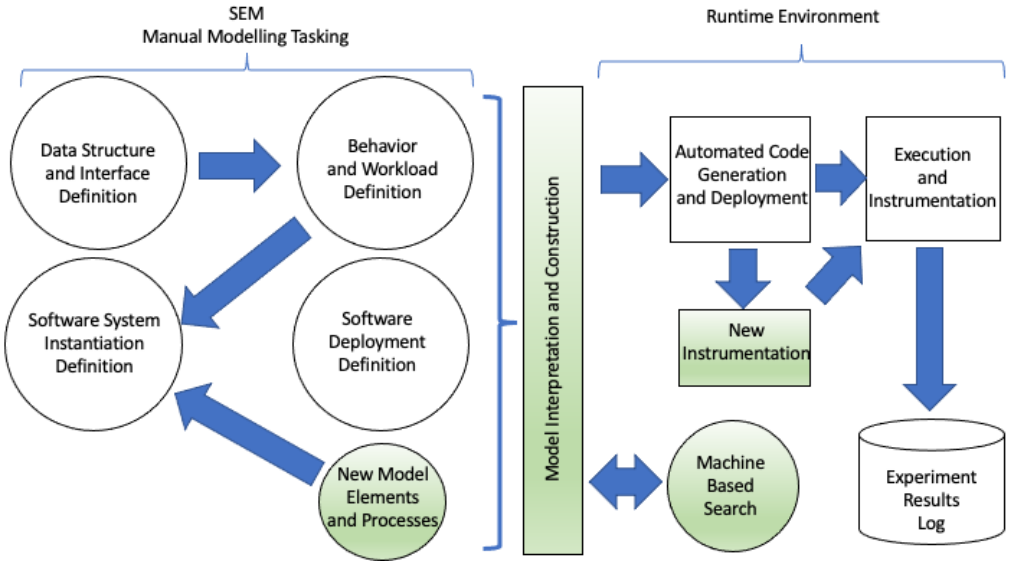

Figure 4.2: Generic high-level SEM execution flow additional components forming the new capability behaviour

of the MEDEA SEM environment (Falkner et al., 2018). As a result, the MEDEA modelling platform was chosen as the foundation from which to develop the new software deployment optimisation capability introduced in this thesis.

Figure 4.2 shows the generic high-level SEM execution flow with the new automated software deployment capability components (coloured in green) introduced in this thesis.

4.1.1 MEDEA Modelling Environment

In the MEDEA modelling process, the SoS design is modelled from various perspectives, including the architecture and workload behaviour, and the assembly of the SoS. These modelling perspectives are designed to capture the system

designer's current understanding of the structure and behaviour of the SoS and can be transitioned through to integration of new systems, adaptation of systems, or changes to the SoS integration approach.

The MEDEA model is defined using a platform-independent modelling language, GraphML,[1] where the definition supports modelling of interfaces, behaviours, and workloads in a manner independent of any intended execution platform. A more detailed illustration of the MEDEA modelling methodology is seen in Figure 4.3, which shows the modelling process, alongside the execution and performance analysis and evaluation processes critical to the methodology. This thesis will introduce enhancements to the modelling, file generation, instrumentation, and evaluation processes (indicated by red stars).

The MEDEA modelling methodology and environment is divided into four modelling aspects that are used to define the system interfaces, system behaviours and workloads, the assembly of the SoS, and deployment of the SoS to the representative test-bed platforms for experimentation and performance prediction (Figure 4.4).

An overview has been provided for each of the four modelling aspects as background detail to support the development of an understanding of the MEDEA modelling approach and environment. Screenshots have also been included (Figure 4.4, 4.5, 4.6, 4.7 and 4.8) to show each modelling context and model elements used to define the system design.

[1] http://graphml.graphdrawing.org

Modelling system architecture

The first aspect of modelling the system design is to model the high-level architecture of the system, represented through the components' interfaces and aggregated data structures (Figure 4.4).

Figure 4.5 illustrates the interface definition of an example system, consisting of various aggregate messages, with some structures exposed. Figure 4.5 also illustrates the interface definition of an example system, consisting of various aggregate messages, with some structures exposed. It also shows the software components and their interfaces that have been defined and will be used to construct the system architecture. The system components are designed to exchange messages using event ports consisting of out-going event ports (shown as the green port icon), and in-going event ports (shown as red port icons).

Modelling system workload

After modelling the system structure, the behaviour and workload for each system needs to be modelled, enabling performance models to be incorporated for each system according to its core behaviours. For each component defined within the system's corresponding interface definition view, a definition for its behaviour and workload is required. Behaviour is modelled either through initiation via a periodic event, or the receipt of an event through a designated event port, which supports independent operation and communication between system sub-components and between systems in the SoS. Within a model, branching constructs support

constraint-based decision-making, resulting in different workloads to being executed according to the environmental conditions.

Figure 4.6 illustrates the behaviour and workload definitions for one of the components within an example system. In this example, the behaviour of the component is initiated by the periodic sending of a message, indicated via a periodic event timer, which then induces the consumption of CPU and memory resources. Upon the completion of the for-loop associated with the periodic events, an out-going message is then generated and transmitted out of the output port.

The model illustrated also shows the logging event post the transmission of a message out of the output port.

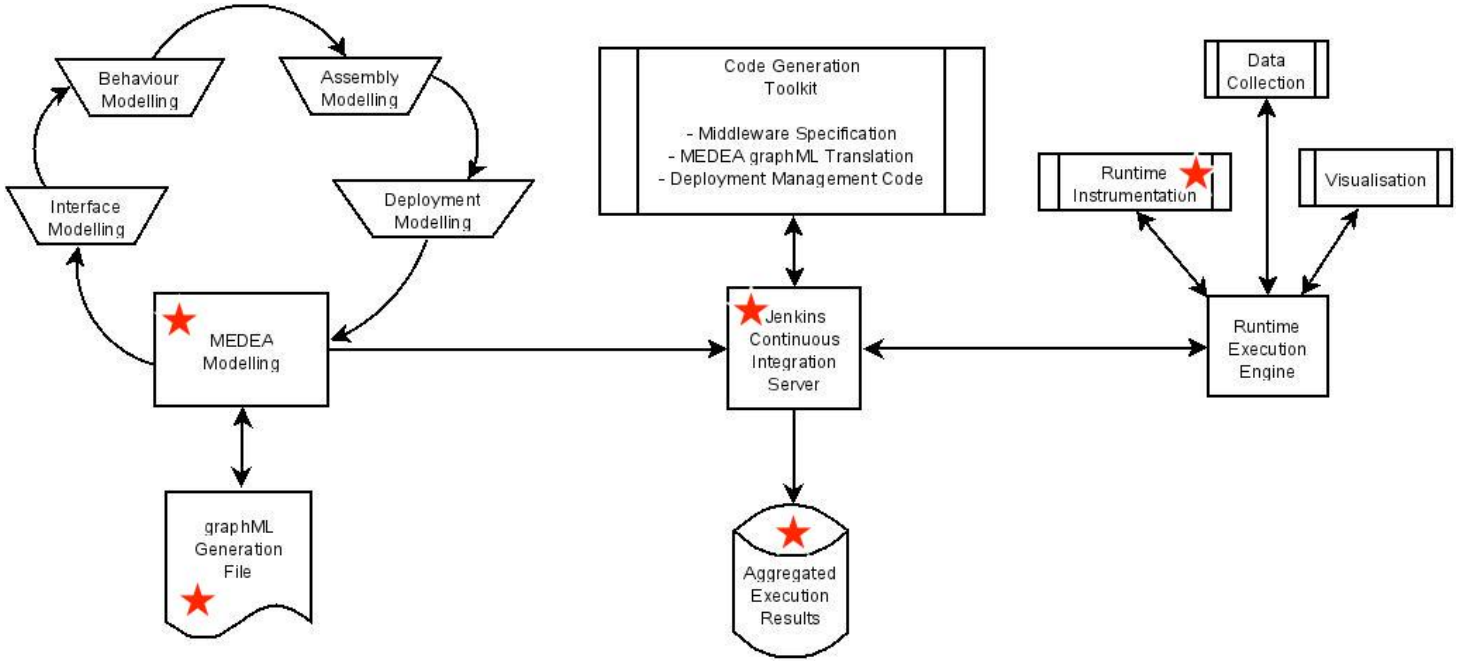

Figure 4.3: The MEDEA performance modelling and prediction process flow diagram (red star indicates process to be enhanced through this research)

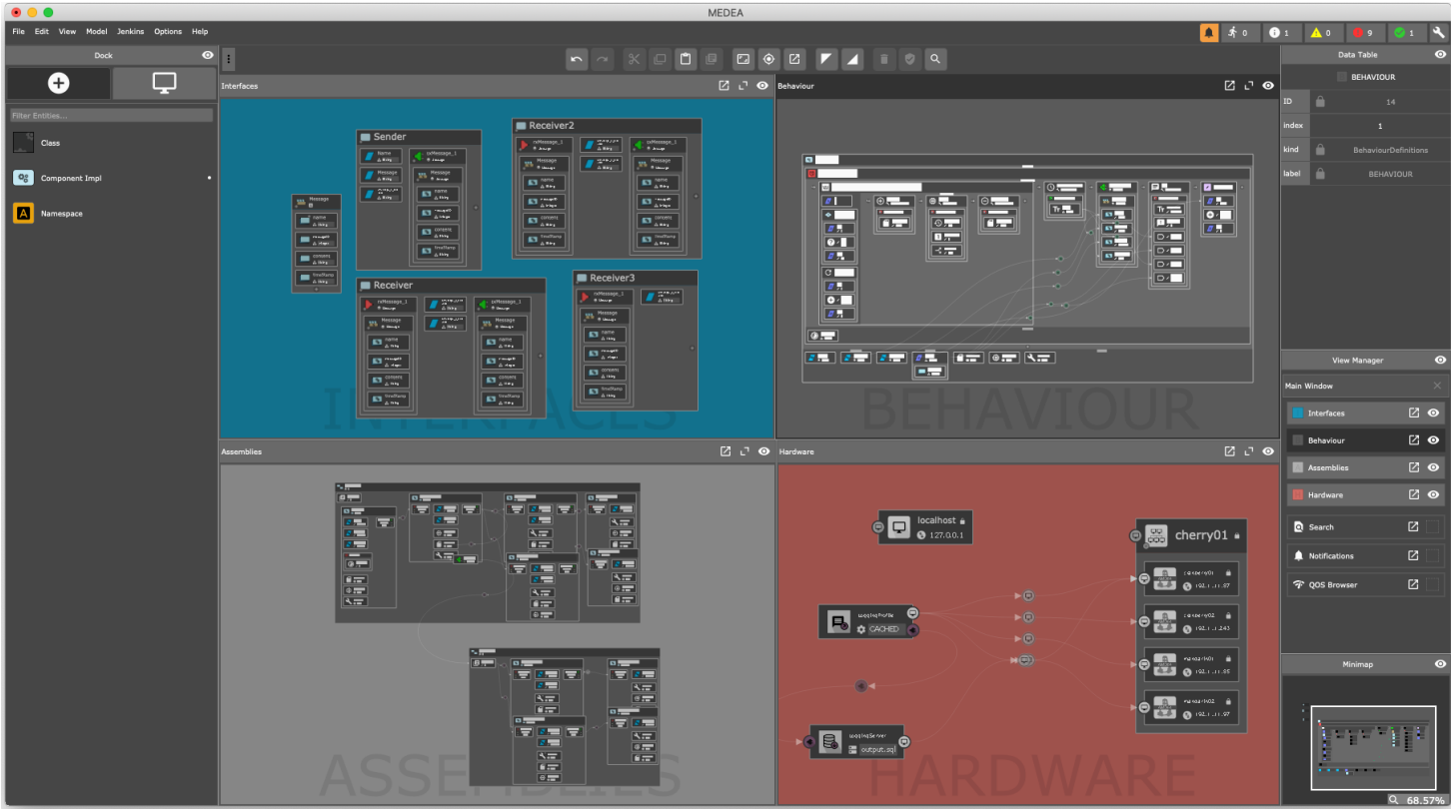

Figure 4.4: The MEDEA modelling environment

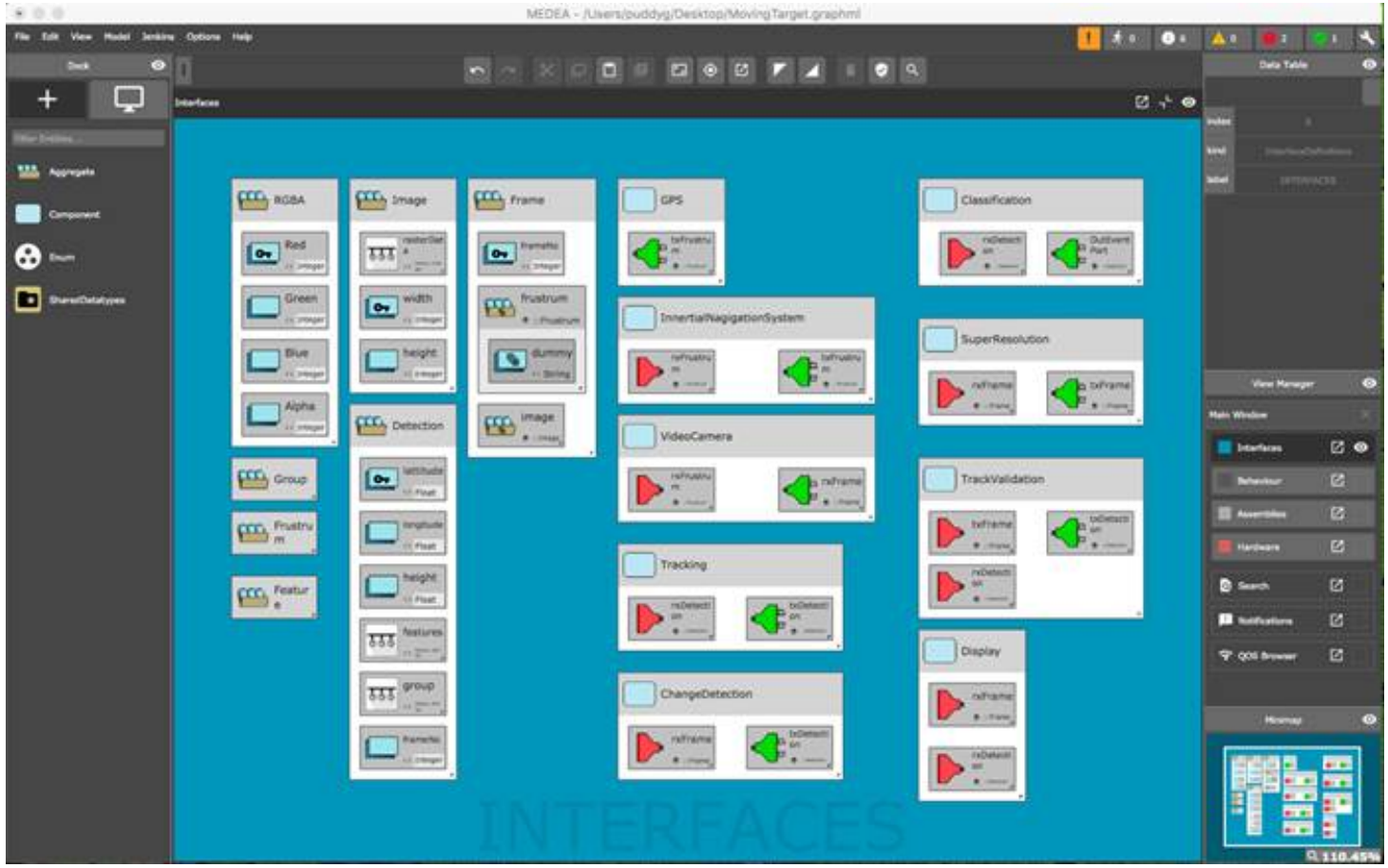

Figure 4.5: MEDEA interface definition modelling

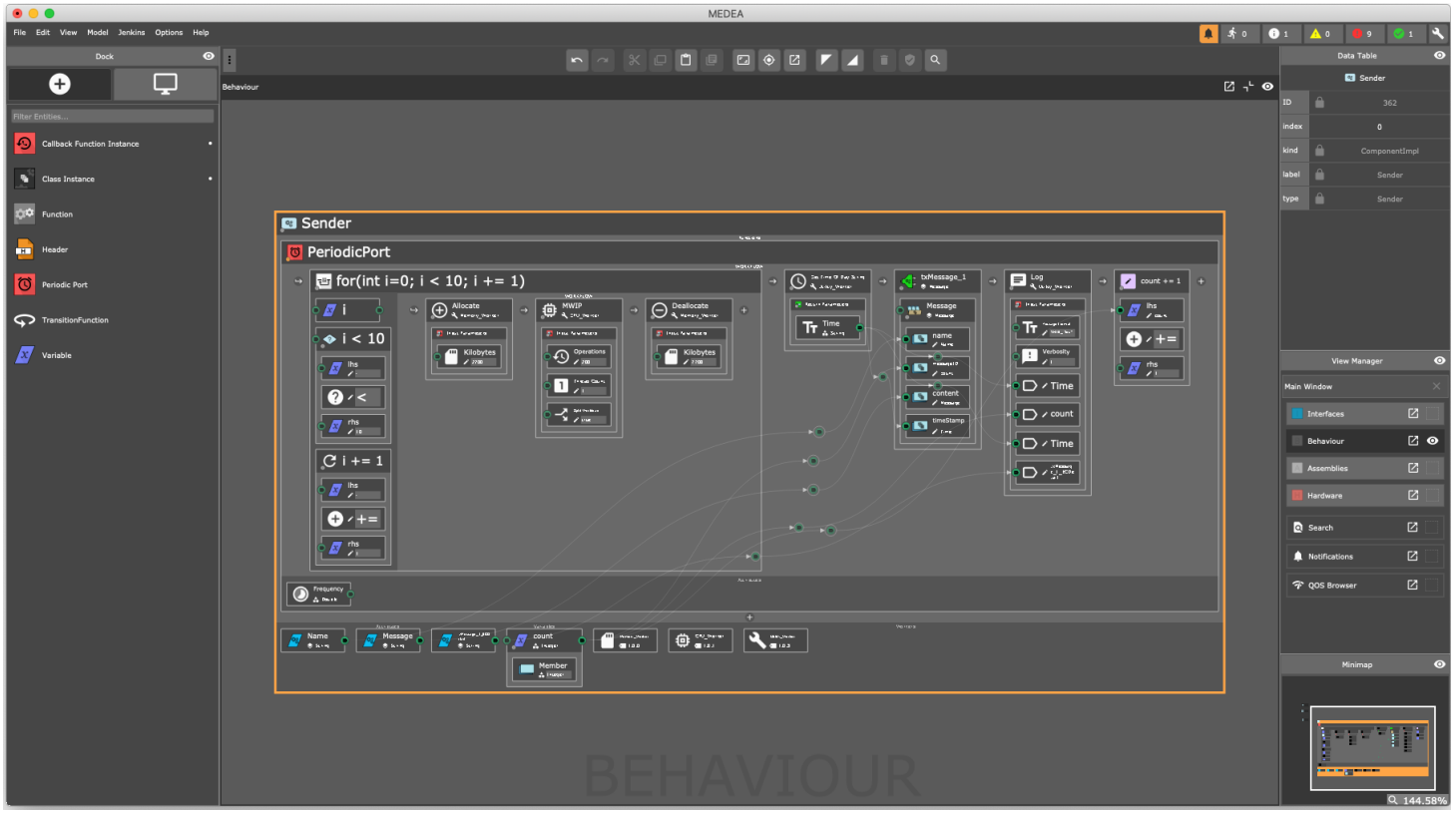

Figure 4.6: MEDEA behaviour and workload definition modelling

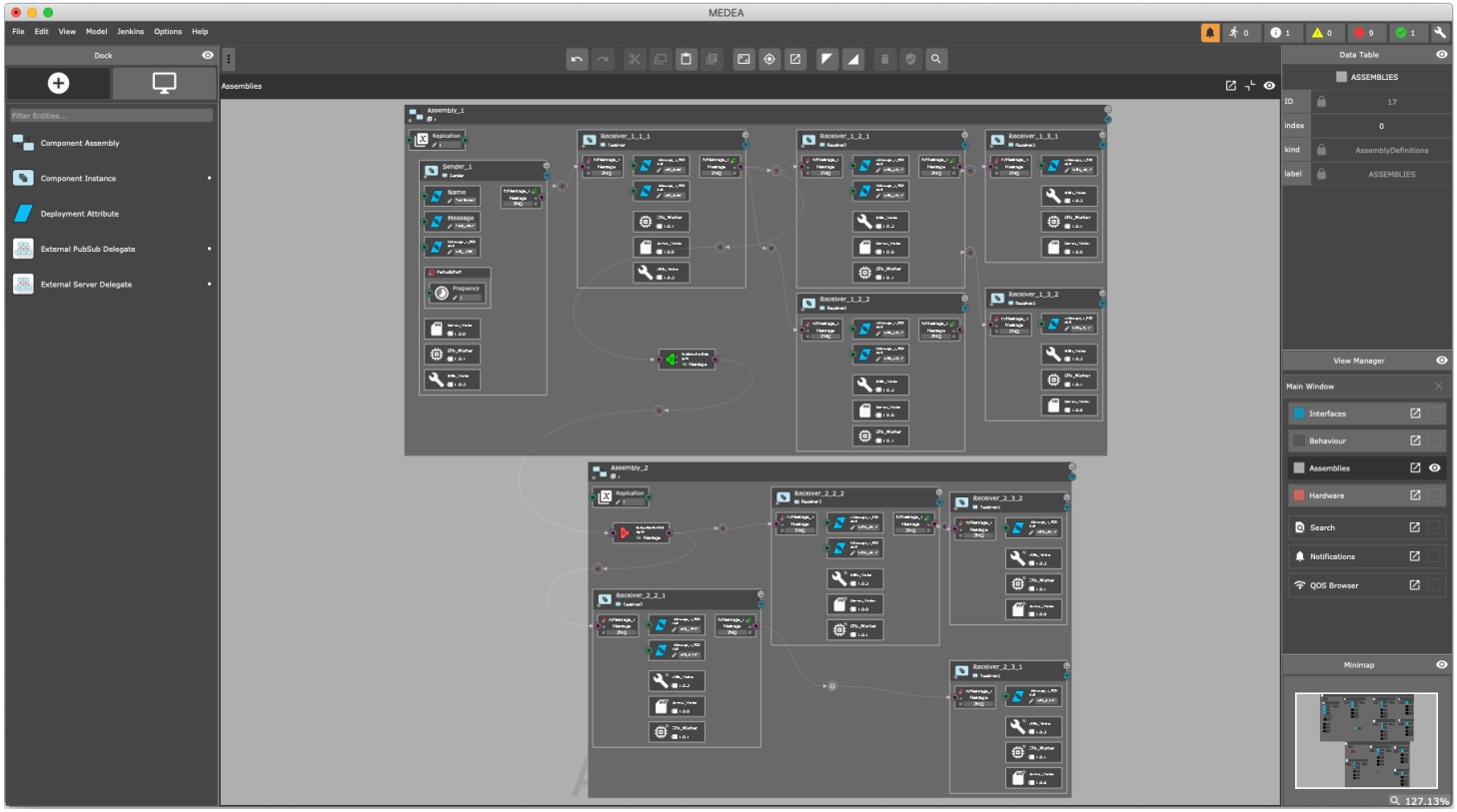

Figure 4.7: MEDEA assembly modelling

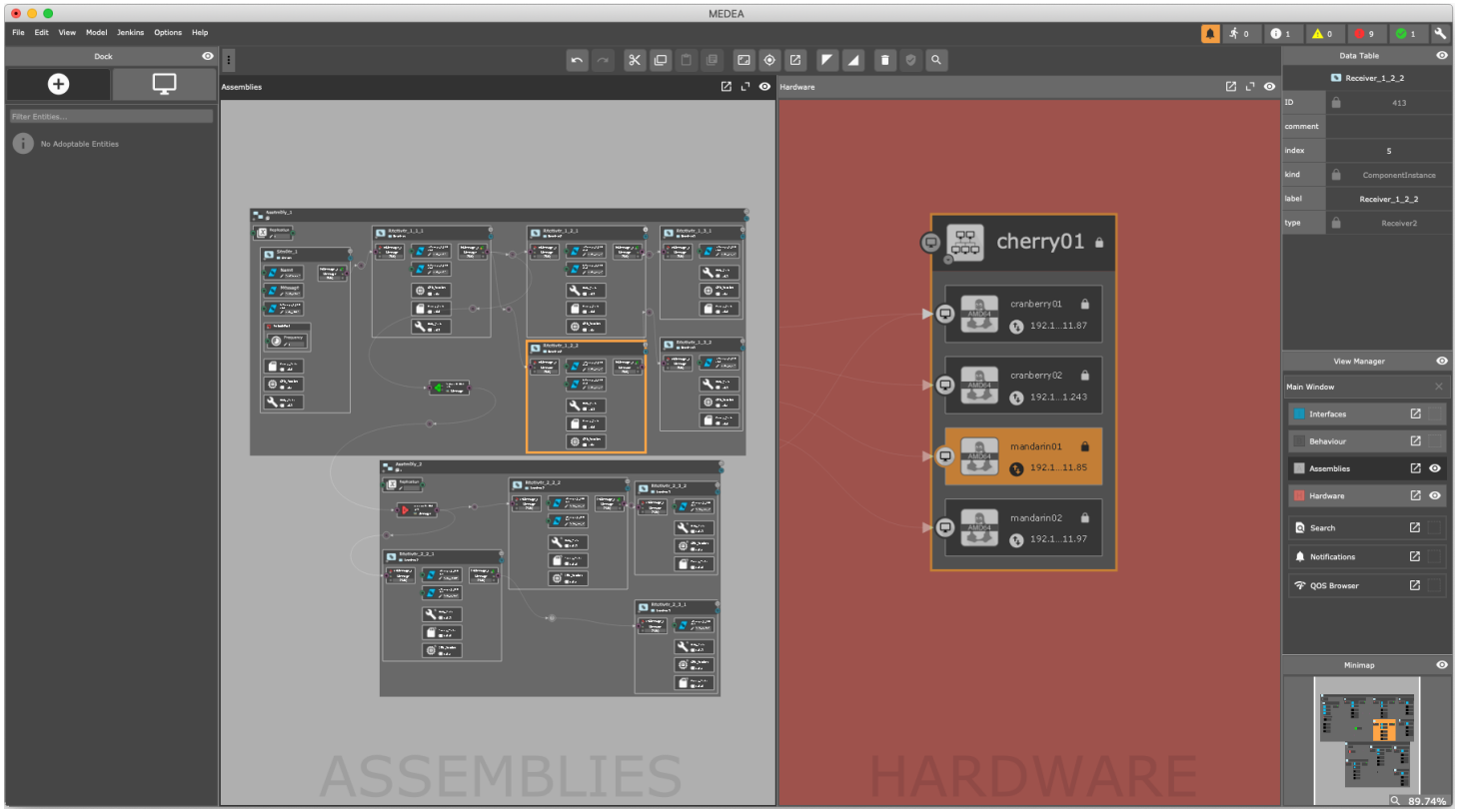

Figure 4.8: MEDEA deployment modelling

Assembly Modelling

The behavioural and workload models defined previously form a generic pattern from which multiple component instances may be created; each following the same behaviour but operating independently. These component instances can then be connected or grouped within assemblies, where external connections between assemblies also occurs. The instantiation of these connected assemblies then delivers the notion of SoS design modelling.

Figure 4.7 illustrates an example system consisting of multiple component instances, including all required connections. In this example, instances of sender and receiver components are created (along with visibility of the instantiated data elements) and explicit connections are made between the in-going and out-going event ports, accordingly. This example also shows a representation of the system connections, via the modelling of in-going and outgoing event port delegates between assemblies, resulting in an SoS model.

Deployment Modelling

The final stage in the MEDEA modelling methodology is the modelling of how these system designs (made up of modelled assemblies and components) will be deployed to the available hardware test-bed. Using a manual modelling process, the system modeller identifies what assembly or component is to be deployed to a particular computing node within the hardware test-bed. Figure 4.8 illustrates this where the 'Receiver_3_2_2' software component has been chosen (highlighted

with the orange border) to be deployed to the computing node 'mandann01' (also highlighted) within the hardware test-bed. This process repeats for each component or assembly needing deployment.

A more detailed description on the MEDEA SEM environment can be found in the Falkner et al., 2018 publication.

Optimised and Automated MEDEA Deployment Modelling

To address the high-level architecture changes presented above, a combination of evolutionary computation, new model elements and modelling processes, and instrumentation will complement the MEDEA modelling and execution environment. This new technique will break the MEDEA execution flow and inject capability to allow for MEDEA model augmentation. This augmentation includes the high-level non-functional design information, optimisation processing and automated code generation that result in construction of MEDEA-compliant models ready for integration back into the MEDEA execution flow and eventual experimentation. Finally, the constructed optimal models will also provide the new instrumentation probe points to be utilised by the new instrumentation analysis framework, which is also interfaced with the MEDEA results database.

Figure 4.9 shows the MEDEA execution flow and the injection points for the new capability being generated with this research. The colouring of parts of the diagram indicate that the additional elements are part of the same capability area of the MEDEA execution flow.

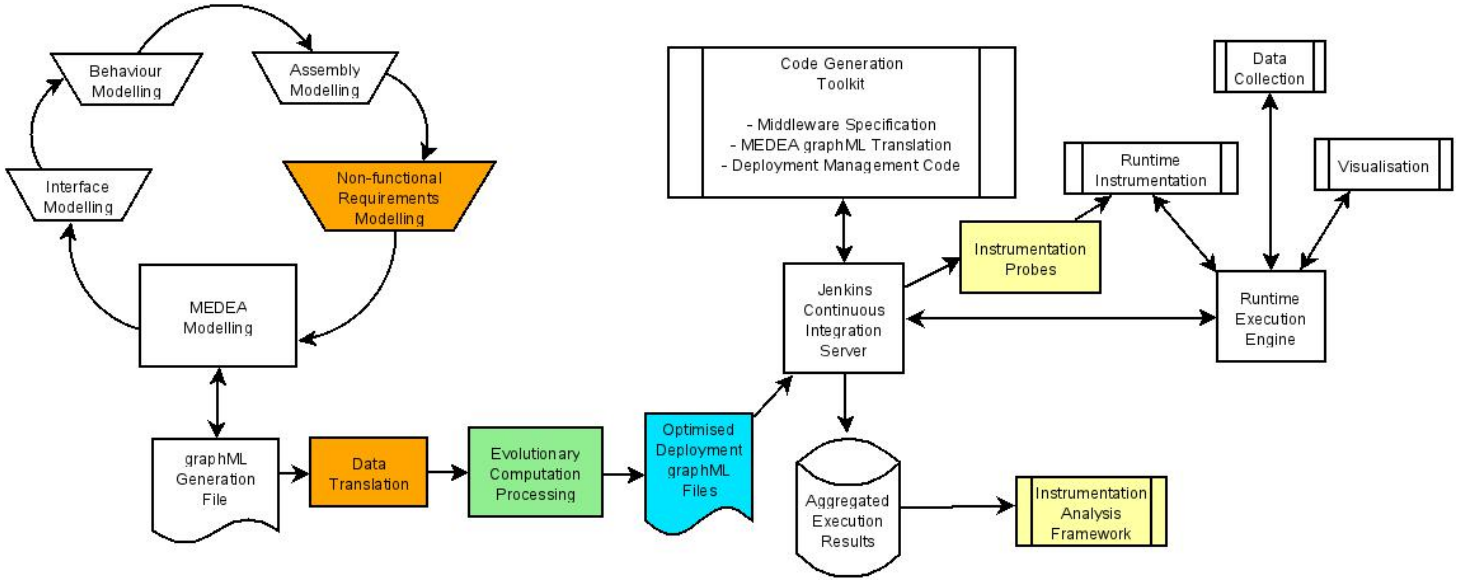

Figure 4.9: MEDEA Flow Diagram with Additional Deployment Optimisation Flow

4.2 System Design Evolution Modelling Methodology

The second part of this thesis introduces a new modelling framework to augment the newly-developed optimal software deployment search capability. With the introduction of this new additional modelling framework, the overall modelling and performance analysis capability is able to consider possible system design evolution and performance impacts for the particular systems (including constrained systems) under investigation.

As found with the SEM approach, the new modelling approach will be based on an MDE philosophy to allow for easy transition between system design modelling views to characterise the system evolution. This will then form as an overlay on the initial system design model and enables the design evolution search.

The architectural philosophy used with this new capability was not to integrate within the SEM-based modelling and analysis capability, but to utilise clear interfaces between environments. As a result, impacts from changes or upgrades to each of the modelling schema associated with each would be minimised.

The development of the new modelling framework encompassed the creation of a new DSML through the use of a meta-meta-modelling environment, the application of the new DSML to construct models, a DSML-to-DSML translation layer and finally an automated model construction capability.

Figure 4.10 shows the high-level design for interfacing the new modelling framework with the SEM modelling context views.

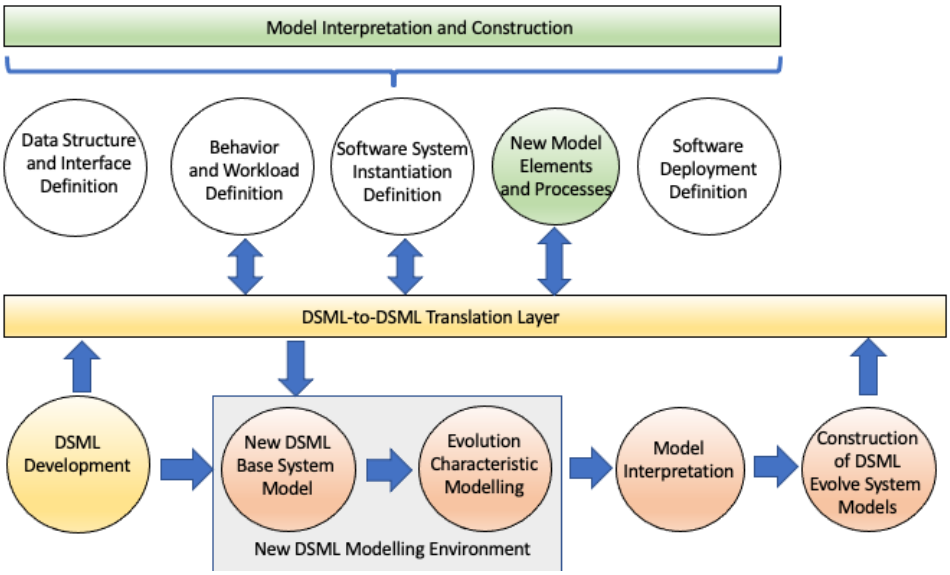

Figure 4.10: New Modelling Framework interfacing with Generic high-level SEM execution flow with additional components behaviour

To enable the construction of the new DSML the meta-meta-modelling language development environment Generic Modelling Environment (GME)[2] was chosen. In addition to this environment enabling the development of the new DSML, it also served as the platform from which to execute the new language and construction of the DSML models.

In addition to using the GME environment for DSML development and utilisation, the CUTS SEM environment (Slaby et al., 2006) was also used for the research effort. The driver behind this was that, at the time of conducting this new research, the broader SEM research program this thesis was part of was using the CUTS environment as a foundation for its research program.

[2]https:www.isis.vanderbilt.eduProjectsgme

Unfortunately, the broader SEM research program moved on from the use of the CUTS platform to developing its own new SEM environment, MEDEA (introduced in the previous section). As a result, all the interfacing, searches, interpretations and code construction capability developed for the CUTS DSML-to-DSML translation layer for this research task was not transferable to the MEDEA environment. Large portions of code became obsolete, requiring a second development effort.

In future research efforts, a fully integrated and executing capability would be realised for the entire new modelling and analysis execution approach. This thesis will introduce this new evolution modelling and analysis capability and detail how it would execute, as well as the expected outcomes.

CUTS Modelling Environment

In line with the generic SEM architecture, the CUTS environment also allows for the definition of the system through modelling context views of structure, behaviour and workload, instantiation and deployment. While MEDEA presents these views in separate modelling canvases, CUTS present a single canvas with modelling traversing through an explorer-like window.

Figure 4.11 shows the CUTS modelling environment with the single modelling canvas (Position Marker '1') along with the explorer window for moving through the model context views, called the 'GME Browser'. Within this browser, the modeller can define the system assembly of instantiated software components (Position Marker '2'), the plans for how to deploy the software system (Position Marker '3'), the definition of the behaviours and workloads (Position Marker '4')

and, finally, the data and software component structure (Position Marker ’5’).

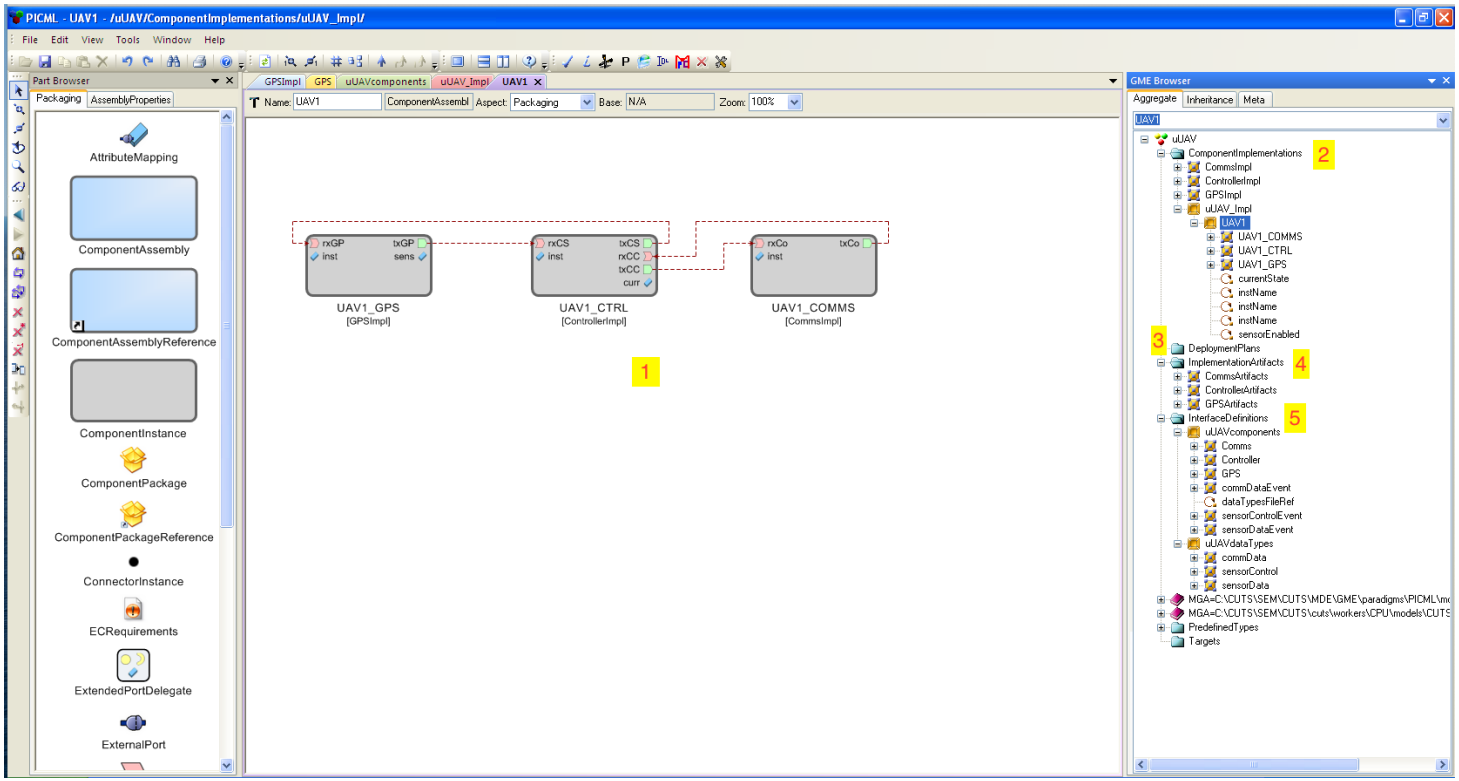

Figure 4.11: CUTS modelling environment with main canvas view and browser window

5. Deployment Optimisation

This chapter will provide technical details on the new approach introduced by this thesis for identifying software deployment solutions that will deliver optimised performance for known non-functional design aspects. The details included are the new modelling elements and methods integrated into the MEDEA modelling environment, followed by the developed method for information extraction to construct an interim information model, and then the usage of that model. It will then introduce the Evolutionary Search and Objective scoring algorithms developed, the optimised MEDEA model construction process, and the new instrumentation and analysis frameworks added to the MEDEA modelling and run-time environments.

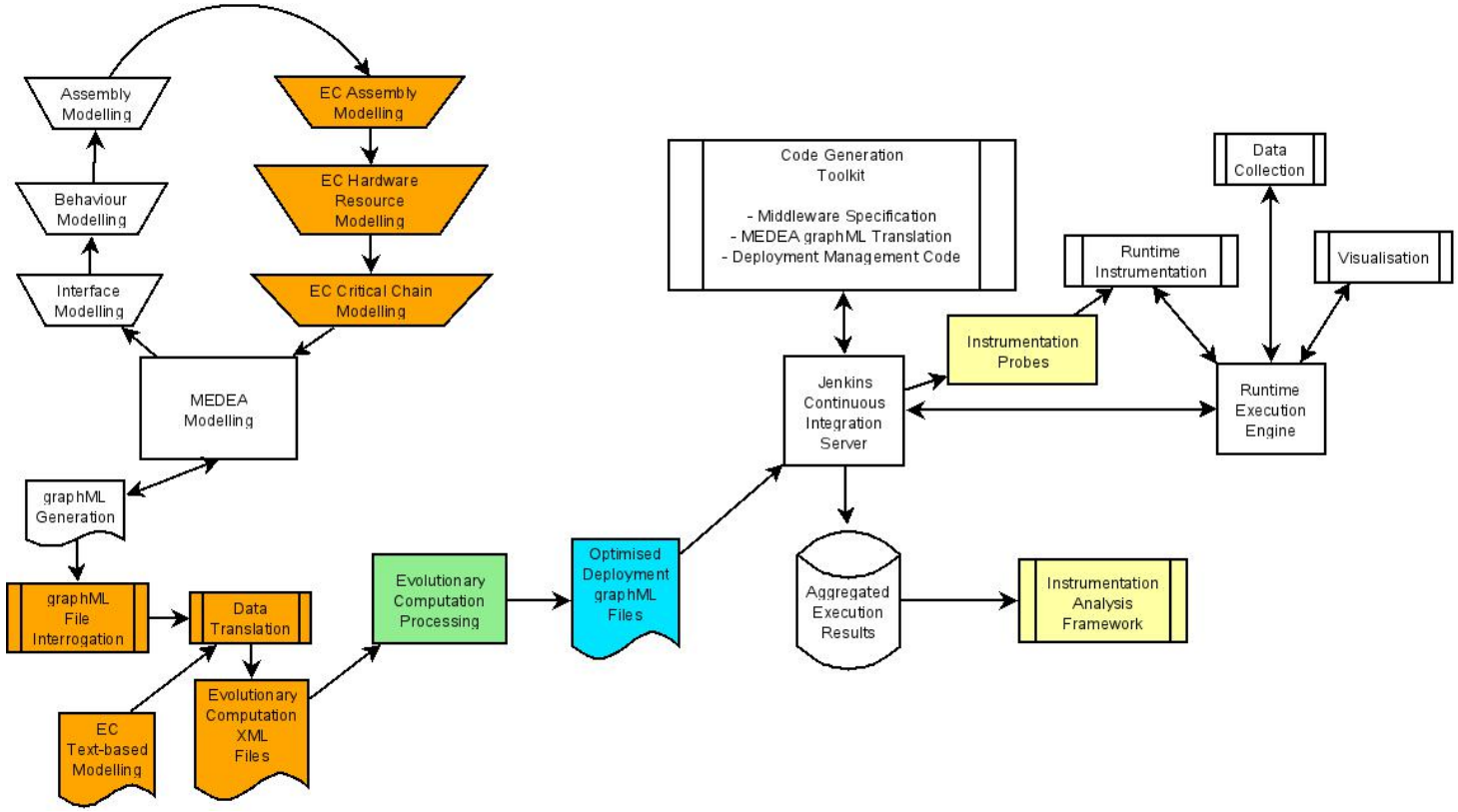

Figure 5.1: MEDEA Flow Diagram with more detailed information modelling process flow

5.1 Information Modelling

As depicted in Figure 4.9, the deployment optimisation flow for this new approach is made up of four components: the information modelling element (highlighted in orange), the evolutionary computation optimisation algorithm (highlighted in green), the graphML file generation (highlighted in blue), and instrumentation analysis (highlighted in yellow).

As established in the previous chapters, the introduced software deployment optimisation is looking to address how to deploy software to best utilise computing resources present within the system computing environment, whilst meeting certain non-functional requirements or constraints. Although many non-functional requirements or constraints could be introduced to drive the deployment optimisation search task, the focus of this introduced approach was centred on the application of constrained SoS, such as for undersea sensor systems. Therefore, to address the many key attributes of an undersea sensor system identified by Akyildiz, Pompili, and Melodia, 2004, Awan et al., 2019, Heidemann et al., 2006 and Ovaliadis, Savage, and Kanakaris, 2010, the approach identifies software deployment options based on temporal performance and Space, Weight and Power (SWaP) requirements, along with mandated deployment constraints. Furthermore, as conditions change over the SoS's life time, the approach can be re-run with changed bounds or design drivers to validate the existing deployment options still satisfy, or develop new deployment options.

The entry point for this new method is the process of capturing these require-

ments and constraints within the context of the system design model. To achieve this, the new method introduces both pictorial and text modelling to satisfy the requirements of the evolutionary computation algorithms introduced in the next section. Figure 5.1 shows an expanded view of the information modelling component flow within the overall MEDEA deployment optimisation flow.

A detailed listing of the non-functional requirements and constraints that feed into the deployment optimisation search process can be found in Appendix C.1.

5.1.1 Graphical Non-Functional Requirement and Constraint Modelling

The graphical modelling approach used in this section is based on the use of a modified MEDEA environment with new modelling elements and processes, and reduced MEDEA system design definition modelling along with MEDEA's graphML file format.

A standard graphML MEDEA model is imported into the new Non-Functional MEDEA modelling environment where definitions for Interfaces, Behaviour and Assemblies of the system design are considered. While definitions for software deployment are not considered, the information present on computing nodes available for deployment is still extracted from the imported graphML file.

The new modelling mechanisms developed for the new Non-functional MEDEA Modelling environment are a combination new modelling elements and annotations of newly-constructed attributes for modelling elements. In the case of the new

modelling elements, these are introduced within the Assemblies context modelling window. Known as *CriticalChain* modelling, the new approach allows for the modeller to identify a critical thread within the system design to be the focus of the investigation. This thread traces through each identified software component and their interfaces, both externally between components and internally between their identified interfaces. An example of the creation of a *CriticalChain* definition within a system design model can be found in Figure 5.2.

To enable the annotation of new attributes to modelling elements within the new Non-functional MEDEA environment, a new entry creation mechanism was introduced for model element data tables. This enables the new attributes to be inserted into the graphML file format for use later down the execution flow.

While this new data table entry mechanism allows for open creation of new data types, restrictions were also introduced. The new capability restricted entries to a set of predefined entries, which were anchored to the requirements of the Evolutionary Computation algorithms. A listing of these pre-defined data entry options can be found in Appendix A.1.

In addition to data entry restriction, two restrictions were introduced to the modelling process. The first was associated with the Behaviour modelling context window, where every interface found within the system design model requires a single workload only associated with it alone. The second of these restrictions was associated with the Assemblies modelling context window and middleware definition to only allow for ZeroMQ middleware alone.

The introduction of these restrictions was not based on any architectural or

capability reasoning, but were only introduced to reduce development complexities for this initial concept. Future development efforts would naturally consider the addition of various middleware options and multiple workloads as part of investigating heterogeneous system designs.

Assemblies Modelling

The Assemblies modelling context window is where the majority of the non-functional modelling occurs to support the optimisation software deployment options search process later on. The focus of the modelling within this context is the identification of the critical design thread or *CriticalChain*, temporal performance requirements and mandated deployment constraints.

The *CriticalChain* definition requires the modeller to trace out an end-to-end thread of the system design. Once defined, this enables end-to-end latency calculations, which becomes the core component of the optimisation Objective Score algorithm. It also serves as the foundation for determining other temporal performance requirements.

Once the *CriticalChain* has been mapped out, other system design details are extracted by tracing back through interfaces, workloads, software components and assemblies associated with that thread. Figure 5.3 shows this arrangement with the yellow *CriticalChain* model entities mapped across the system's components and assembly models via the interface or delegate modelling elements (highlighted).

The second step to the *CriticalChain* mapping effort is the selection of certain segments of the mapped thread, and the addition of predefined Data Key values,

as required. As a result, additional non-functional requirements can be attached to parts of the critical thread to allow for high-level requirements (such as latency requirements) to be included into the optimisation search.

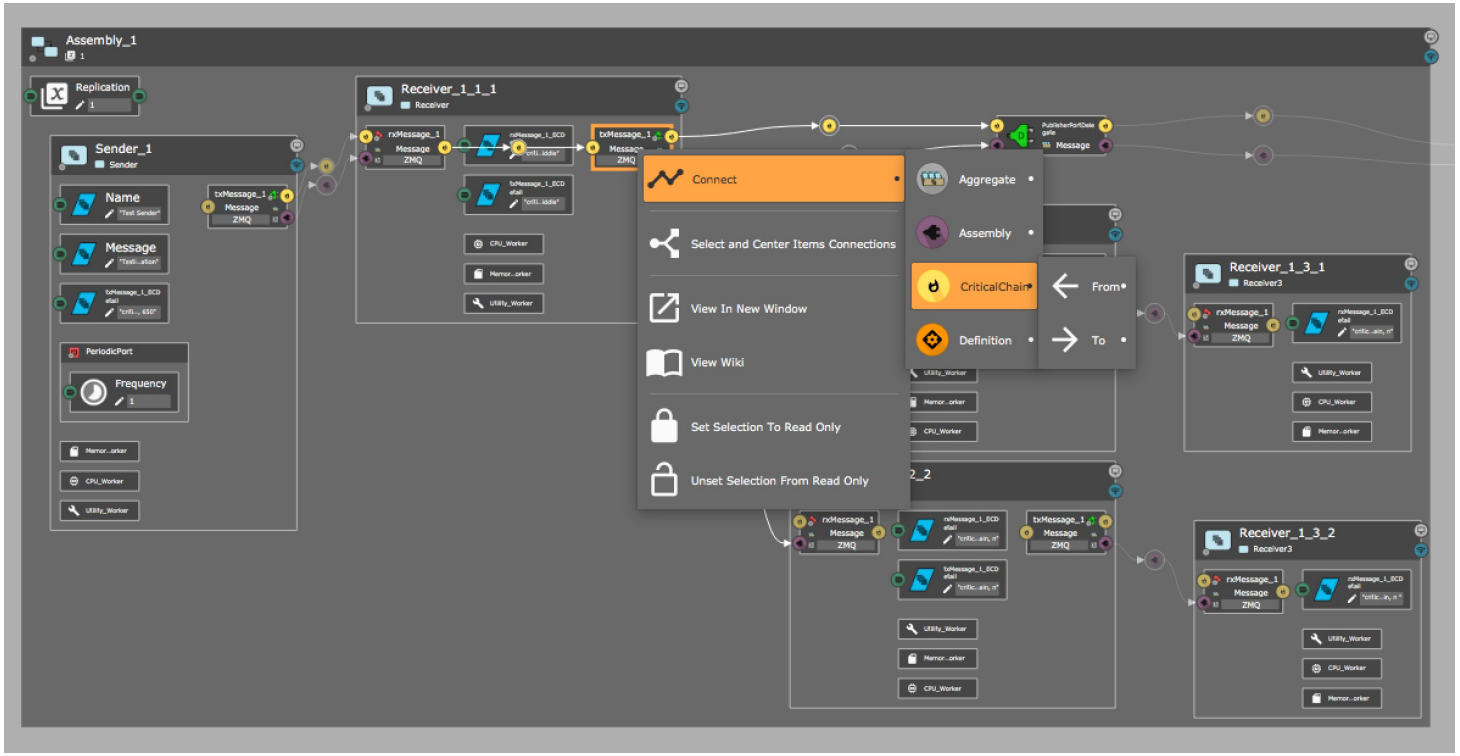

Figure 5.2: CriticalChain thread definition within a MEDEA system design model

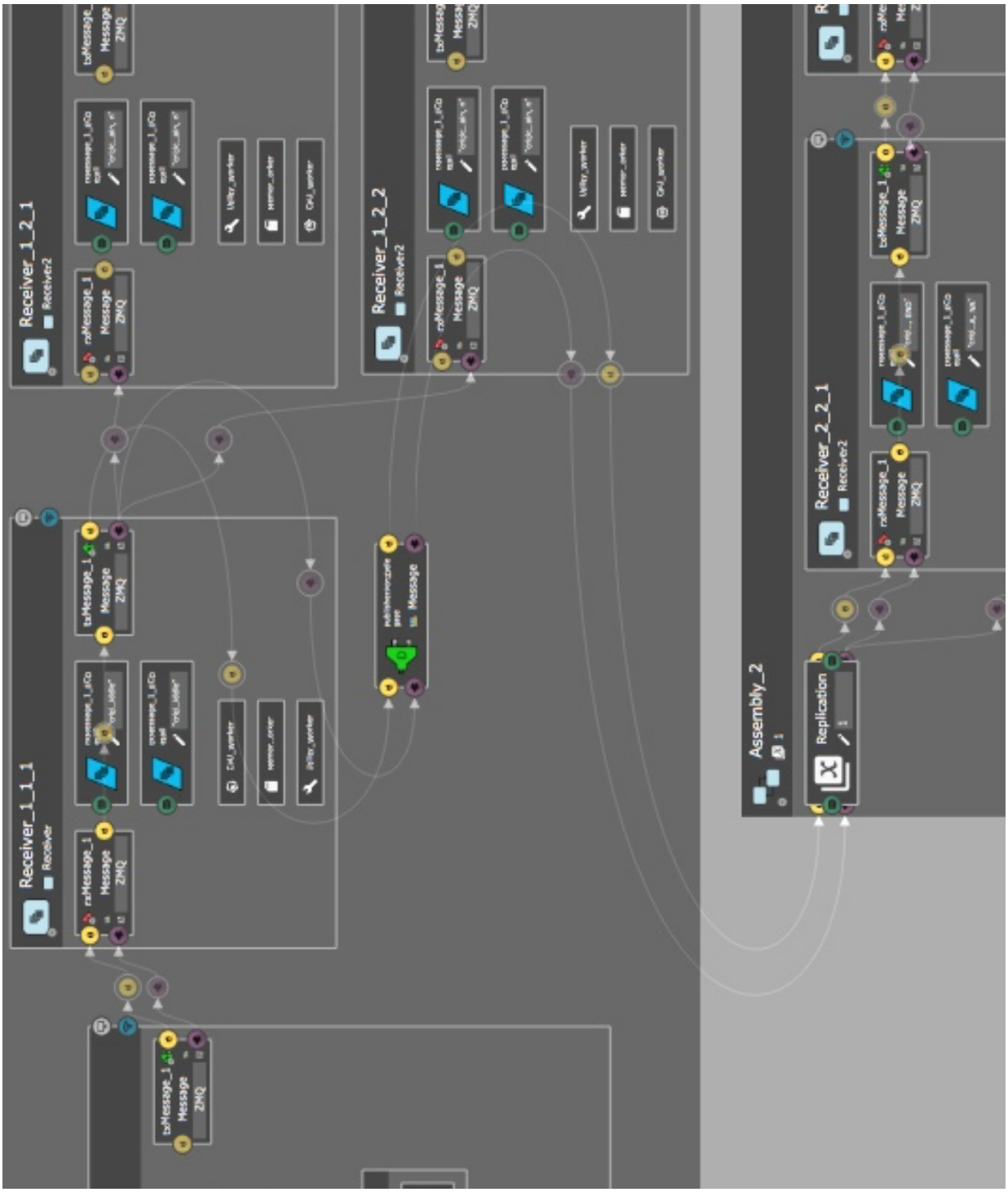

Figure 5.3: MEDEA Critical Chain mapping within a cropped view of the Assemblies Modelling Context

In Figure 5.4 we can see a section of *CriticalChain* mapping for a system design under investigation, with a segment highlighted for the purpose of attaching additional attributes within its data table. In this case, the highlighted segment is a single connection between two software component interfaces.

The *internalConnection* Data Key allows the modeller to define whether this connection is between components or within, therefore dictating whether the other required Data Table pre-defined attributes are required. If applicable, the modeller can then define a maximum latency performance requirement for the identified connection, as well as whether or not this connection is part of a string of interfaces that must satisfy a total maximum latency performance requirement.

Upon completion of the *CriticalChain* modelling, the modelling then moves to attaching predefined Data Table attributes to existing standard MEDEA modelling elements.

Details about these allowable attributes for new or existing MEDEA modelling elements can be found in Appendix A.1.

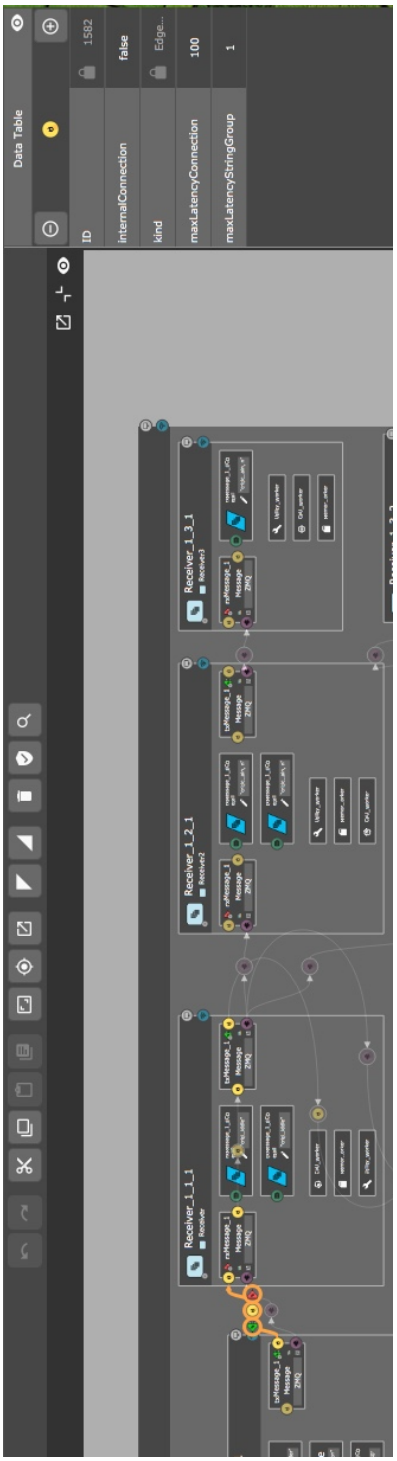

Figure 5.4: MEDEA Assemblies Context Modelling cropped view with attribute information

Hardware Modelling

As described in Chapter 4 the hardware modelling context window considers the available computing environment and how software components or assemblies deploy to the available nodes. For this introduced deployment optimisation process, the original MEDEA graphML file does not include deployment profile details for either software components or assemblies mapping to the available computing resources. The construction of these deployment profiles and insertion into MEDEA graphML files occurs as part of the outcomes from the optimisation process.

Within the hardware modelling context window, no new modelling elements are found, but rather the modeller attaches predefined attributes to identified hardware modelling elements. Figure 5.5 shows an example of a computing environment with predefined attributes attached to a computing node.

As detailed in Appendix A.1, the hardware modelling context allows for weight attributes associated with CPU and memory resources utilisation. The attachment of these weights occurs on a per individual computing node basis and allows the modeller to define the importance to adhering to those non-functional constraints for the particular computing node.

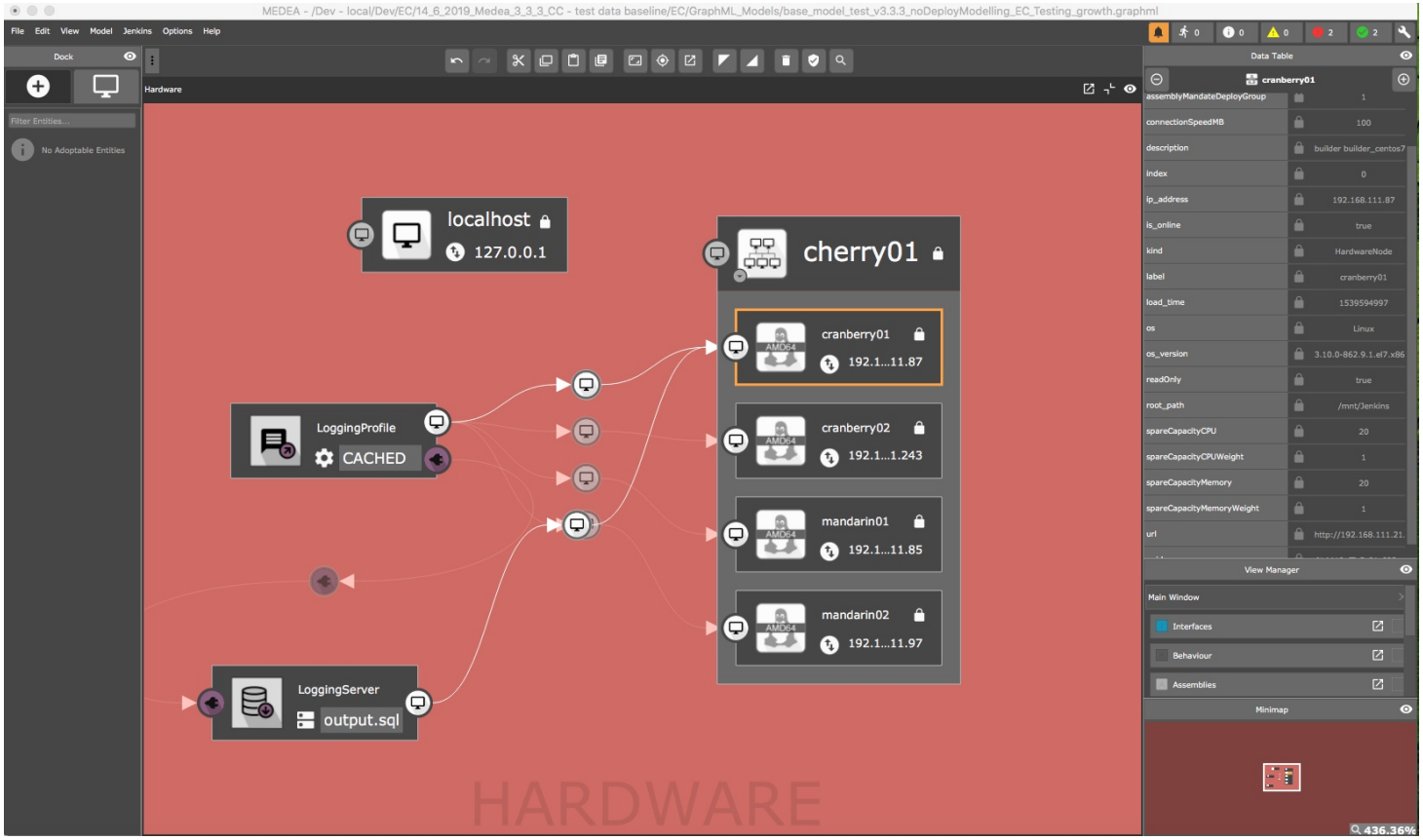

Figure 5.5: MEDEA Hardware Context: view with attribute information

5.1.2 Text-based Non-Functional Requirements and Constraint Modelling

The text-based component of the Non-Functional modelling phase accounts for three areas of information: the configuration items for the optimisation algorithms, weights for certain algorithms and valid search parameters. These text-based definitions, along with pictorial-based definitions, form the complete set of variables required for the optimisation processing algorithms to execute.

Furthermore, the modeller is also able to define growth factors over a series of optimisation executions for identified workers and their workload levels. As a result, the modeller is able to explore resource growth influences on maintaining optimised outcomes, and resulting software deployment options. This capability also forms the start of a foundation that enables exploration of system evolution via workload growth and its impact on software deployment options for particular computing environments.

All attributes are chosen via a Command Line Interface (CLI). The details of these text-based model elements can be are found in Appendix A.2 and Appendix A.3.

The last area of text-based modelling allows for selection of the type of optimisation algorithm to execute, processing initial conditions and characteristics of the optimisation processing to be executed. The details on the CLI options can also be found in Appendix A.4.

5.2 Evolutionary Computation-based deployment optimisation search

Figure 5.1 shows the Evolutionary Computation (EC) processing function as a single component (highlighted in green) within the additional processing flow for the overall MEDEA processing flow. The figure shows this function utilises the XML files created out of the modelling (pictorial and text-based) phase previously described. Following the completion of its optimisation search, it then feeds into the Optimised Deployment graphML Files (highlighted in blue), also represented as a single component within this execution flow view.

At the core of the EC processing is an Objective Function that utilises details extracted from system design models, as well as using the measured data. Both of these inputs are also used to enable modelling of the system performance. The Objective Function is then executed with local and global search algorithms to produce the search for and identification of optimised deployment solutions for the current system under investigation. Figure 5.6 shows the execution flow and each component is discussed below.

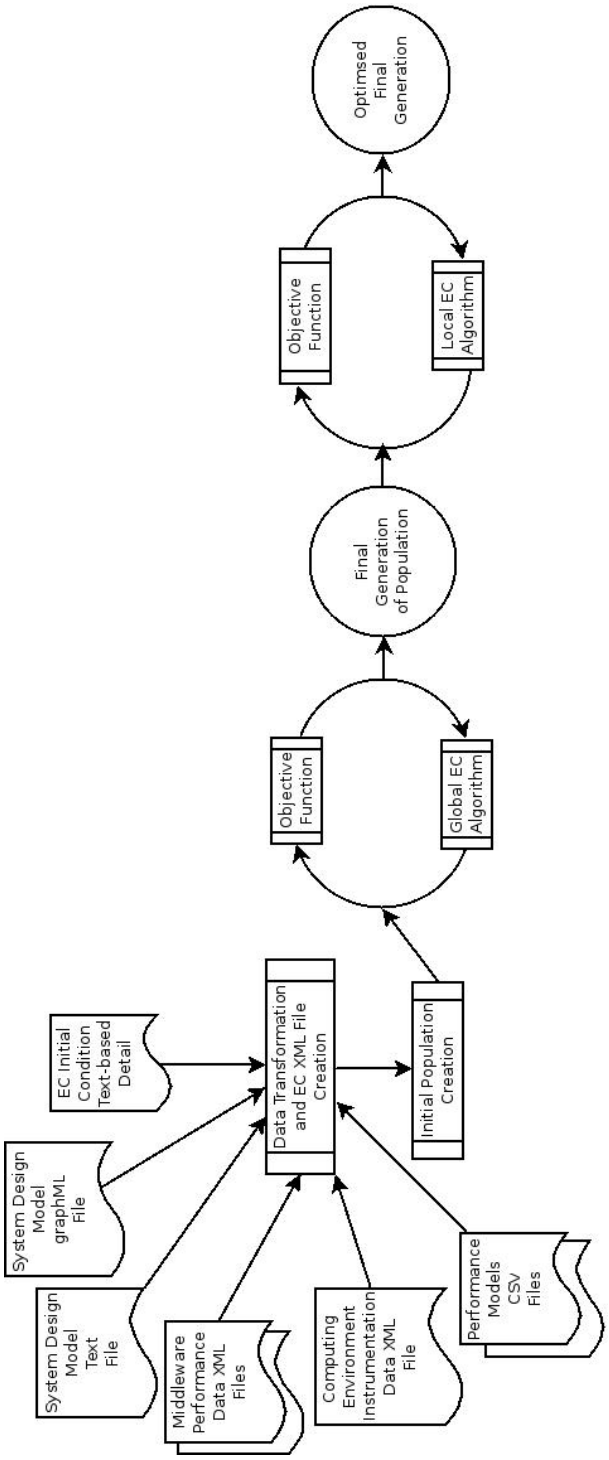

Figure 5.6: Evolutionary Computation Processing Flow

5.2.1 System Design Model

The system design model houses all the information about the computing resource requirements, interface and information flow requirements, the makeup of the system architecture, and details about the computing nodes' availability for deployment within the system's computing environment.

In addition, the model holds the non-function requirements and constraints the software deployment needs to abide by, either completely or partially (depending on the configuration of the search). Finally, the system design model holds the initial condition and search characteristics required for the particular optimisation processing for both local and global population searches.

5.2.2 Computing Environment Instrumented Data

The instrumented data fed into the Objective Function considers resource utilisation of the ZeroMQ [1] middleware and latency performance of the ZeroMQ middleware for particular message sizes. It also details the IP address, chip-set details, and the level of CPU and memory resources available for consumption. For measurement of available CPU resources, the approach taken to bench-mark is based around MEDEA's use of the Whetstone CPU bench-marking (Curnow and Wichmann, 1976), where measurements are in Millions of Whetstone Instructions Per Second (MWIPS).

The instrumentation process occurs via scripts executed as Jenkins [2] jobs, and

[1] https://zeromq.org/
[2] https://jenkins.io/

result in the information being aggregated into an XML file for translation and use further down the optimisation processing flow. An example of this file can be found in Appendix B.1. While ZeroMQ is the focus of the instrumented data future developments would look to explore heterogeneous middleware deployment, as well as the use of this TCP socket.

5.2.3 Performance Models

In a bid to reduce complexities and effort for instrumenting CPU and network utilisation across many scenarios, a resource utilisation model was integrated into the Objective Function processing. While the model used was a simple representation of the relationship between resource utilisation and time delay, it was suitable for this initial concept development.

The model used to represent the impact of resource utilisation for CPU and network resources is based on a simple M/M/1 queuing model. This model determines the time delay multiplier for wait times of queued workloads. Two additional modelling points above the 100% utilisation point are provided to cater for over utilisation scenarios. The demands and raw wait times of the SoS workloads deployed to CPU and network resources for each node are still calculated dynamically, and change with each deployment scenario. The M/M/1 model (with over utilisation threshold points) is a static modelling approach is used to account for further impacts (and delays) as resource utilisation reaches saturation and beyond. Future research would investigate the use of improved models or new instrumentation approaches.

As can be seen in Figure 5.7, up to the 100% utilisation point, the time delay multiplier is a function of resource utilisation, and is based on Little's Law (Little and Graves, 2008) where:

$$L =' Mean \quad Number \quad Of \quad Jobs \quad In \quad System'$$

$$W =' Mean \quad Response \quad Time'$$

and:

$$Arrival \quad rate \quad = \quad \lambda$$

where the 'Mean Number Of Jobs In The System' can be calculated as a function of system utilisation 'ρ':

$$L = \quad \frac{\rho}{1-\rho}$$

then:

$$W = \quad \left(\frac{\rho}{1-\rho}\right)\frac{1}{\lambda}$$

or:

$$W = \quad \frac{\frac{1}{\mu}}{1-\rho}$$

but:

$$'Service \quad Rate' \quad = \mu$$

$$'Service \quad Time' \quad = \frac{1}{\mu}$$

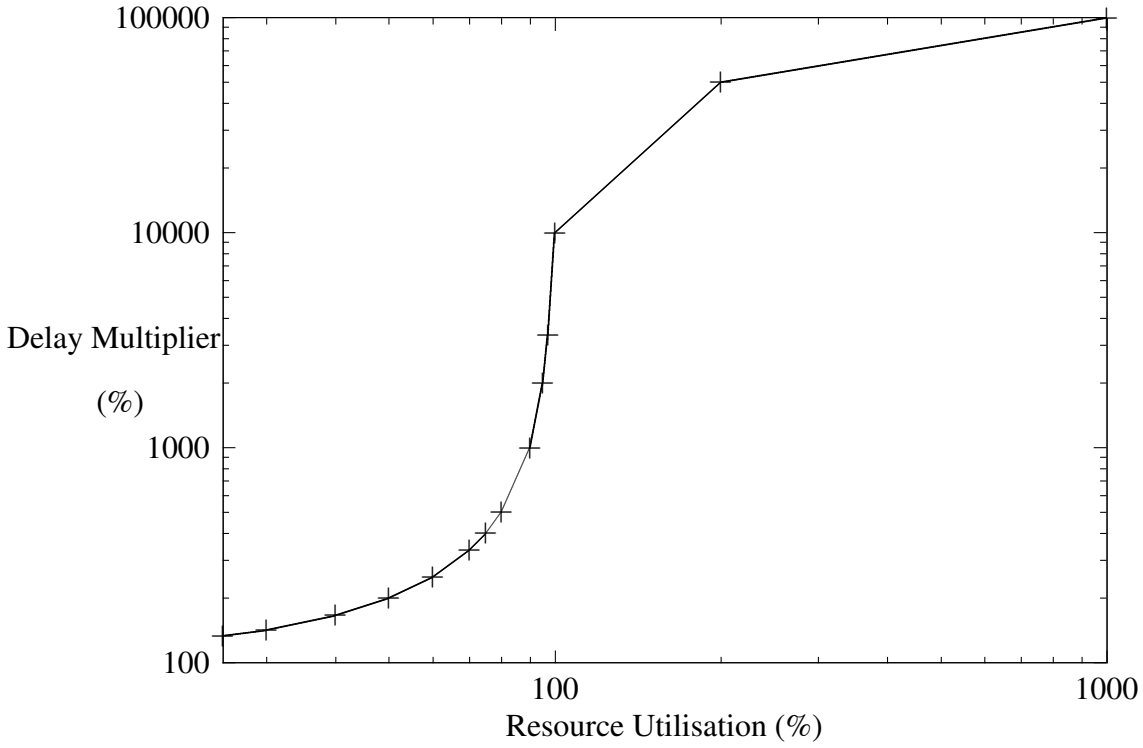

Figure 5.7: Model Characteristic Curve: Resource Utilisation v Time Delay Multiplier

therefore:

$$'Mean \quad Response \quad Time' \quad = \frac{'Service \quad Time'}{1-\rho}$$

The data points used from this model can be seen in the Table 5.1 where a number of utilisation points have been selected. This includes over utilisation points, and their corresponding time delay multiplication data points. These values have been captured in CSV file form and utilised for both CPU and network interface utilisation for each node within the available computing environment. Furthermore, while the granularity of the data points is reasonably large, the

	Resource Utilisation (%)														
	0	25	30	40	50	60	70	75	80	90	95	97	100	200	1000
Time Delay Multiply Factor	100	133	142	166	200	250	333	400	500	1000	2000	3333	10000	50000	100000

Table 5.1: Utilisation values used

optimisation process that make use of this data also contains an interpolation step to allow for improved data fidelity, as required.

5.2.4 Data Transformation

This component of the optimisation execution flows handles the ingestion and transformation of all measured data points, performance model details and system model information into the format required to be used by the optimisation processing algorithms.

As detailed above, a design decision was made to decouple the Deployment Optimisation framework from the underlying MEDEA Modelling environment through the use of an information interface layer. This interface layer is the EC XML file and is the conduit for information into the optimisation process via the translation and aggregation process applied to numerous information artefacts created during the system modelling phases.

The system information artefacts used for information aggregation and creation of the EC XML file are:

- A modified graphML file with additional non-functional performance requirement details,
- A text-based file also containing non-functional performance requirement details,

- An XML file containing details on hardware specifications on all available computing nodes and extracted via a Jenkins job,
- XML files containing performance details on middleware (only on ZeroMQ initially) for each available computing node, used for interpolation purposes within the optimisation process and are not part of the EC XML aggregration file,
- CSV files holding data points from the model used to represent performance impacts based on resource utilisation factors. Files exist for both CPU and Network interface utilisation for each available node. Once again, the data are used for interpolation purposes within the optimisation process and are not part of the EC XML aggregation file,
- A text-based file used to hold optimisation configuration details and initial conditions.

The constructed EC XML file is divided into three main areas within the main *SystemEnvironment* element. These sub-elements are the *GlobalSettings*, *HardwareDetail* and *SoftwareDetail* elements. An example of these sections of the XML file can be found in Appendix B.2, Appendix B.3 and Appendix B.4.

5.2.5 Initial Population

Using the optimisation process configuration and initial condition settings within the EC XML file, the size of the population is set, the required number of generations set, and local optimisation algorithm selected. Once set, the optimisation process can begin the deployment optimisation search with the construction of the

initial population.

The initial population creation process utilises a random selection procedure for attributing a computing node to a software component for deployment purposes. These software components and computing node associations are then captured in an array to represent a deployment profile for all software components found within the system (not just the ones that make up the critical thread). These deployment profiles then become the population members of the initial population.

Once constructed, each deployment profile population member is passed to the Objective Function Algorithm to determine an initial Objective Score and to determine if this initial deployment satisfies any non-functional construct that may be imposed. The execution flow and calculations within the Objective Function processing will be described later in this chapter.

If the deployment profile option is deemed as suitable, it is added to the initial population, otherwise the deployment profile is discarded and the random construction process repeated until a suitable deployment profile is found. This is conducted for each population member created, up to the defined population size where it is then passed to the Evolutionary Computation (EC) local search process (refer to the next section). Figure 5.8 provides an overview of the processing flow for the initial population creation and then entry to the local optimisation search process.

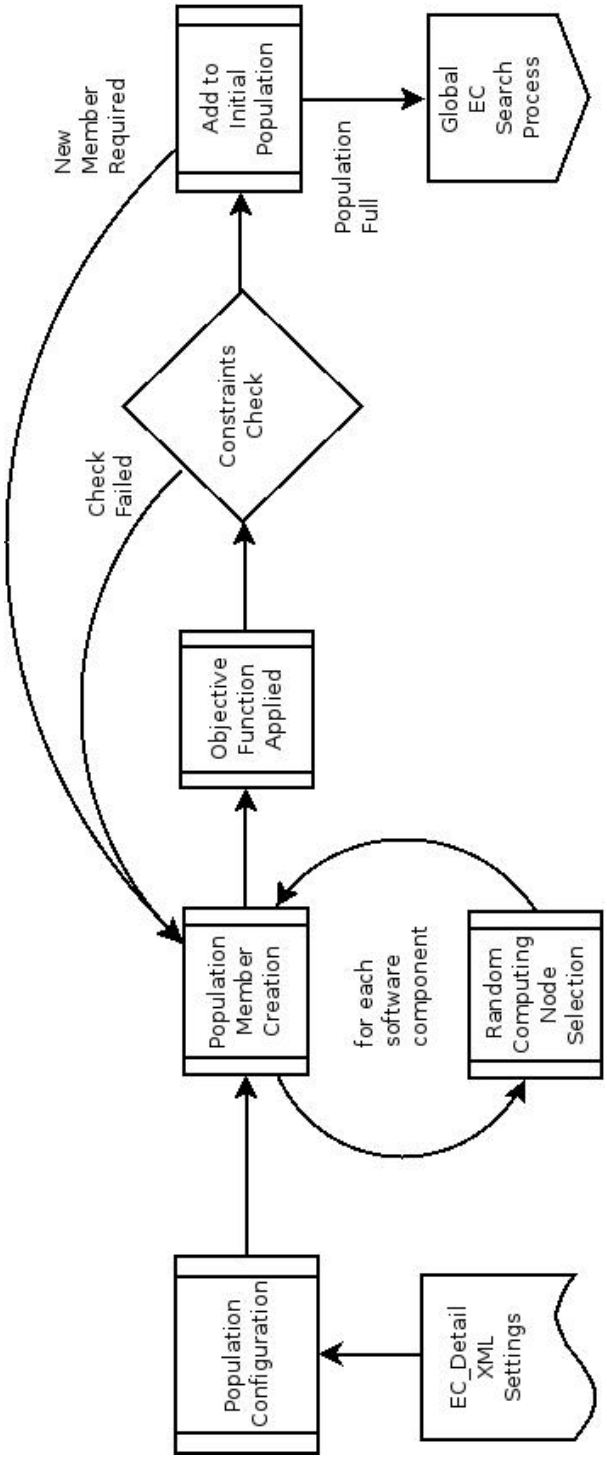

Figure 5.8: Initial Population Creation Processing Flow

5.2.6 Optimisation Search

We employ a hybrid optimisation algorithm, based on the research and observations made on improved performances gained with hybrid optimisation algorithms (Chen et al., 2007; Weise et al., 2016; Jamal, 2019). The algorithm conducts a global EC optimisation search and then applies a local optimisation search to the global search solutions.

Global EC Algorithm

The EC process adopted for the global search was based on the use of Fitness Proportionate Selection (Holland, 1975), or the roulette wheel selection technique, combined with crossover and mutation. These were implemented from scratch to allow for flexibility and better customisation.

With Fitness Proportionate Selection being based around the probability of selection proportional to the size of the Objective Score, the greater the Objective Score, the greater the probability of that search solution (population member) being selected from the population. In this case, the developed optimisation process looks to minimise the Objective Score, so the smaller the Objective Score the better the optimisation solution is for the software deployment profile. Therefore, the Objective Scores produced are modified to ensure the size of the allocation is proportional to the minimised Objective Scores.

The creation of the roulette wheel and its segments are based on determining the total Objective Score of the current population and then calculating the

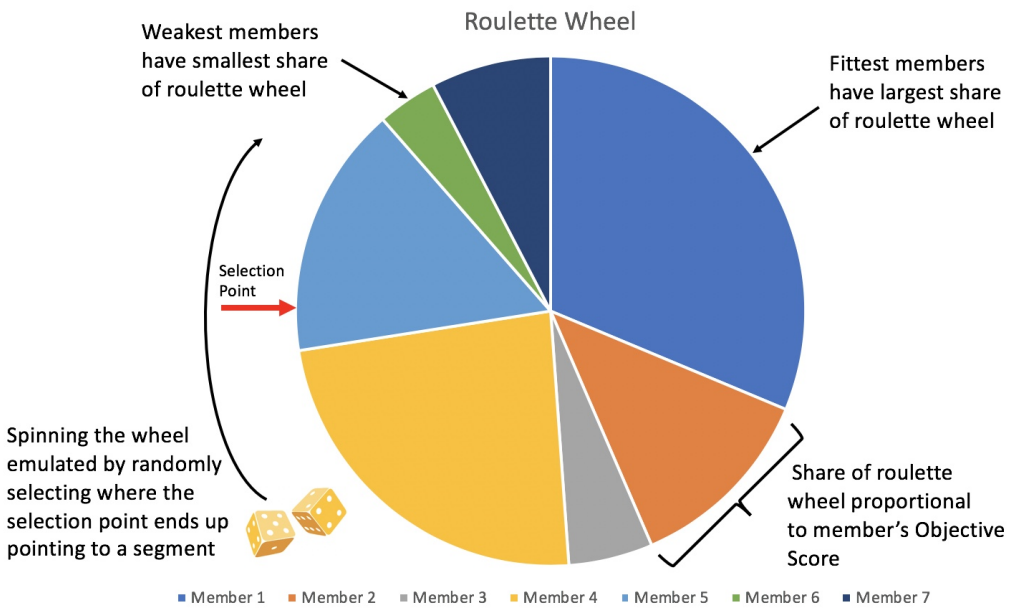

Figure 5.9: Roulette wheel segment allocation and selection

percentage of the current population member's Objective Score compared to the total Objective Score of the population. The percentage size then correlates to the size of the segment (or slice) of the roulette wheel allocated for that population member. Furthermore, the construction of the roulette wheel and proportions are re-calculated with each deployment scenario to be investigated.

The roulette wheel is then used to select the parents by randomly selecting a number between 0 and 100 and choosing the percentage segment of the roulette wheel that aligns with that random number (Figure 5.9). This is repeated for the selection of both parents, and a check is completed to ensure different parents are chosen to create two new children.

Following the selection of the two parents, the optimisation process then

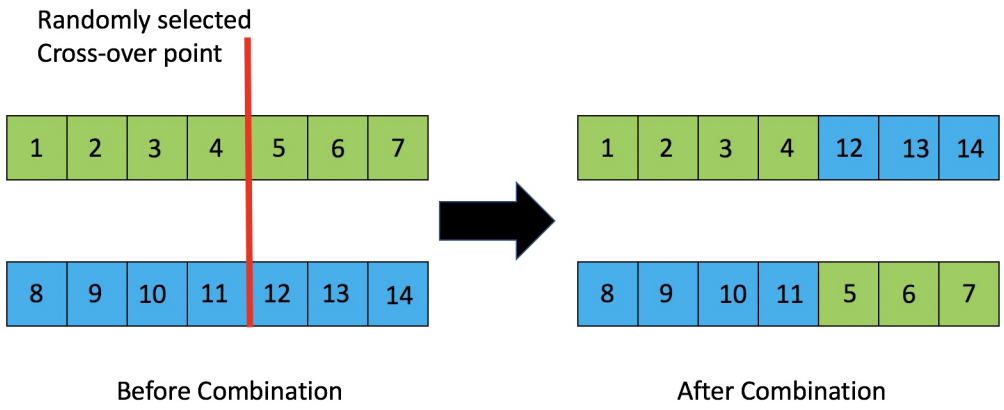

Figure 5.10: Cross-over selection and parent combination

determines the crossover point between the parents to create the children, and the mutation point for each child created.

In the case of the crossover point, this is once again determined with a random number generator, whereby a number is chosen between a range of '0' to the number of the system software component looking to be deployed through the optimisation process. Once determined, a combination of the parents' deployment profile details is created around that point to produce two children (Figure 5.10).

Following the creation of the two children, a mutation process is applied to each child. The mutation process consists of randomly selecting an element from the deployment array of each child, then randomly changing the number within that element. This change process consists of a random selection of a number from a set of numbers representing the nodes available for deployment (Figure 5.11).

The last step in this search process is to confirm the created children satisfy the constraints defined in the System Optimisation Model. This also includes the

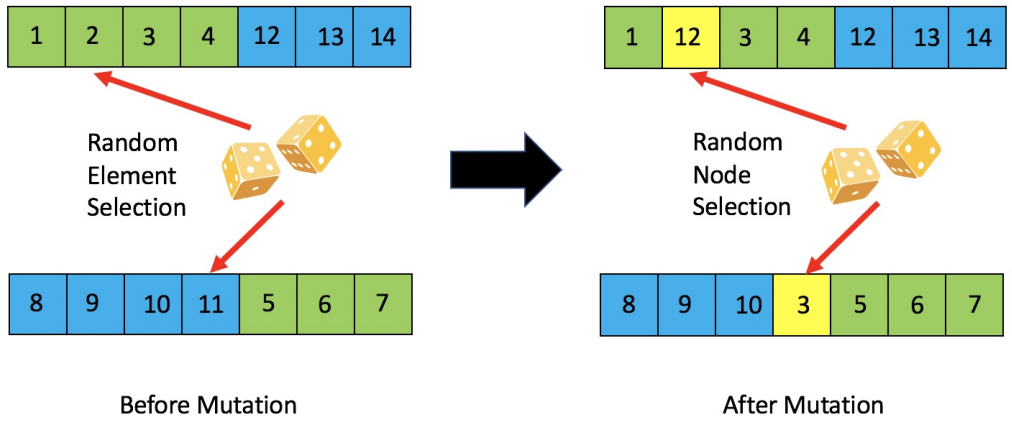

Figure 5.11: Mutation process

calculation of an Objective Score for that child. If a child's deployment profile does not satisfy the constraints, neither child is added to the population that will become the basis for the creation of the next generation. Following the creation of valid children and the complete new generation, a new population is determined by taking the population members with the best Objective Scores up to the population size.

The entire global search process is repeated to create each new generation, while the number of generations to be created is set as part of the System Optimisation Model.

The execution flow for the roulette wheel construction and Objective Score conversion process, as well as the overall execution flow for the Global Search process can be found in Figure 5.12.

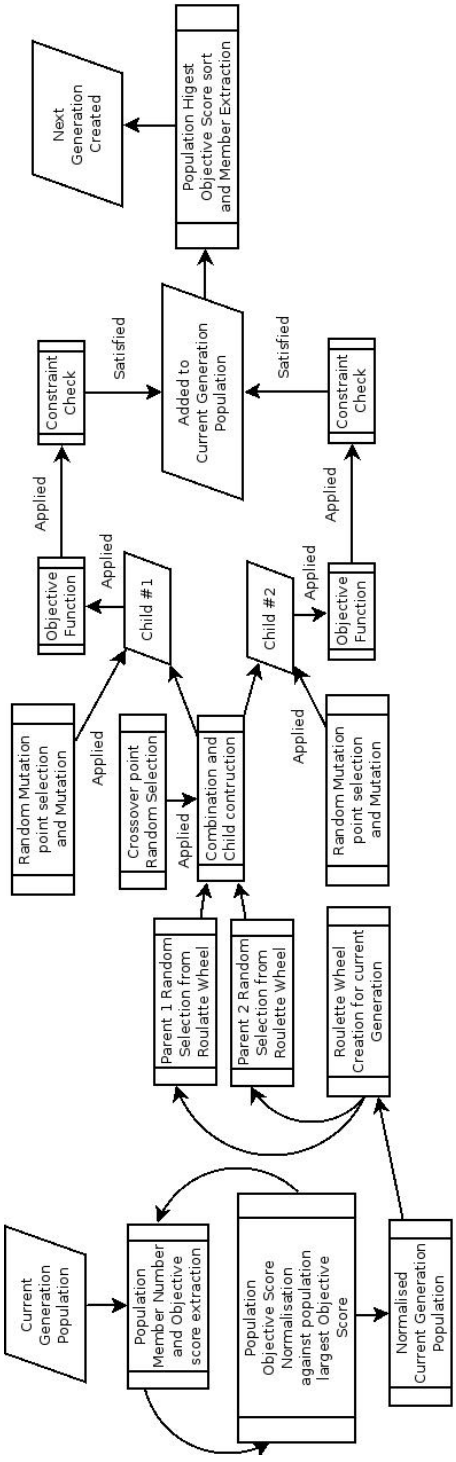

Figure 5.12: Global Search Process Flow

Local EC Algorithm

Following the development of the final generation of the optimal deployment solutions, the population is optimised further through the application of a local search process. For the local search there are three choices of local search algorithms that can be executed. The three search algorithms are *exchange*, *jump* or *inversion* (Sutton, Neumann, and Nallaperuma, 2014).

Each of the local search algorithms undertake an approach of randomly selecting element positions within the solution array (in this case the deployment profile array) and changes the arrangement of the solution array around those position selections. The *exchange* algorithm exchanges the elements within the solution array at positions 'i' and 'j', while the *jump* algorithm moves the element within the solution at position 'i' to position 'j' and then moves the other elements from position 'i + 1' and 'j' (inclusive) by one step towards position 'i'. Finally, the *inversion* algorithm adjusts the solution array by reversing the order of the solution array elements between position 'i' and position 'j'. Figure 5.13 shows the adjustment approaches for each of the local search algorithms.

With each new local search solution array representing the node deployment profile, the Objective Function is then applied to determine the Objective Score. If the resulting Objective Score is greater than the original Objective Score (node deployment profile), the new solution array is changed out with the original solution array and taken into the next local search process step.

The number of local searches conducted is predefined as part of the System Optimisation Model. Figure 5.14 shows the basic steps of the local search process

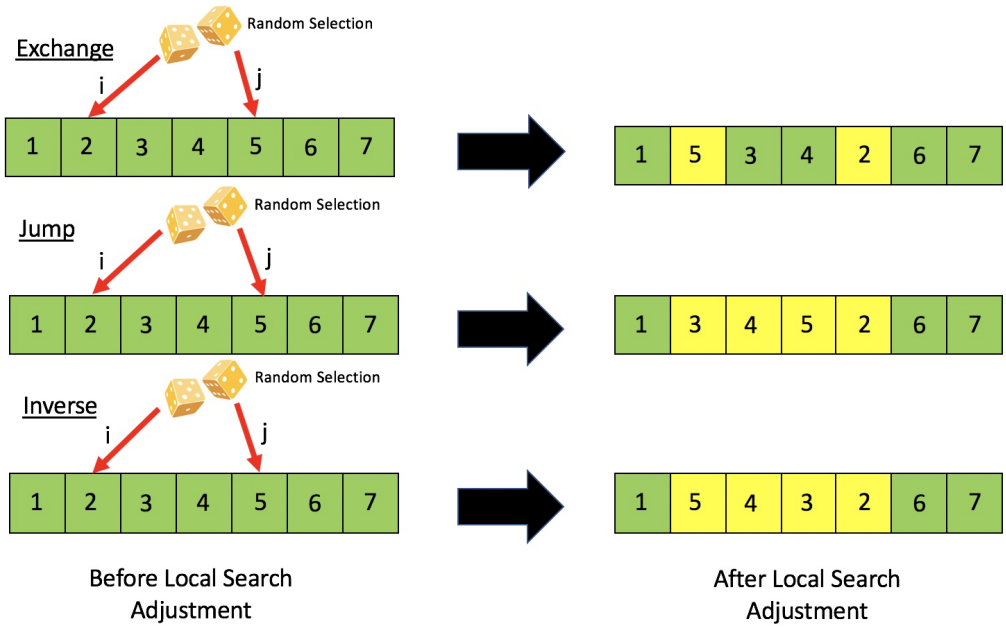

Figure 5.13: Adjustment approaches for local search algorithms

being applied to the final generation's population.

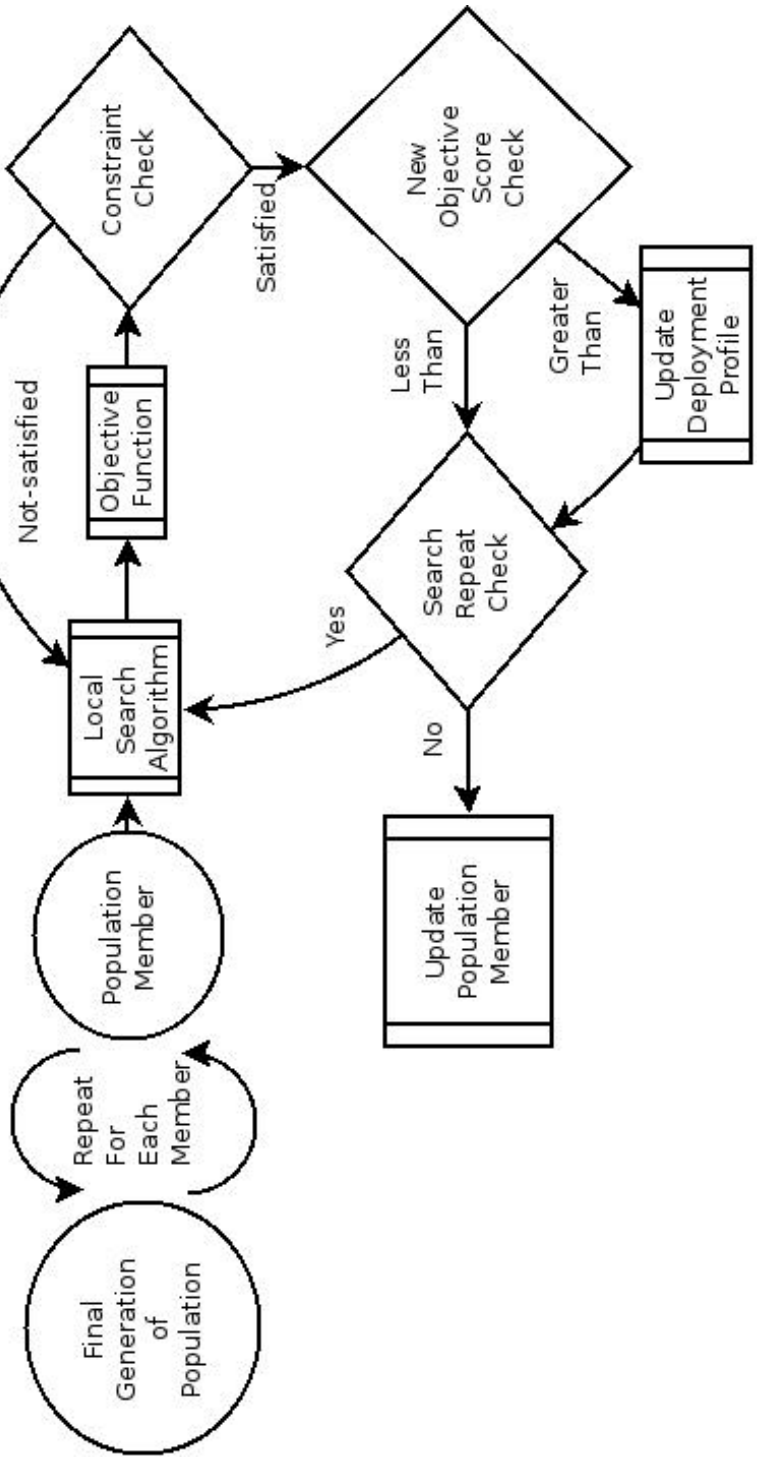

Figure 5.14: Initial Population Local Search Process Flow

Objective Function

The Objective Function has two core components of calculation that determine the score and magnitude of the optimisation fitness for the particular search. The first component is a temporal performance calculation associated with the particular critical thread (CriticalChain) of interest in the system design, traced out as part of the System Optimisation Modelling phase. The second component is the calculation to indicate how well the deployment profile satisfies the non-functional requirements and constraints, also defined as part of the System Optimisation modelling phase.

As described in the modelling section above, a critical design thread is traced out within the system design of interest. Using this defined thread, the end-to-end latency is calculated as part of the search process, with the aim of making the latency as small as possible, while still satisfying non-functional requirements and constraints.

As can be seen in Figure 5.15, the trace considers the software components, the paths and workloads within the software components, and the connections between software components via their interfaces. The trace also details performance requirements associated with those interface connections that are part of the thread. In addition to the various system elements associated with the thread, the trace details latency requirements related to segments of the critical thread.

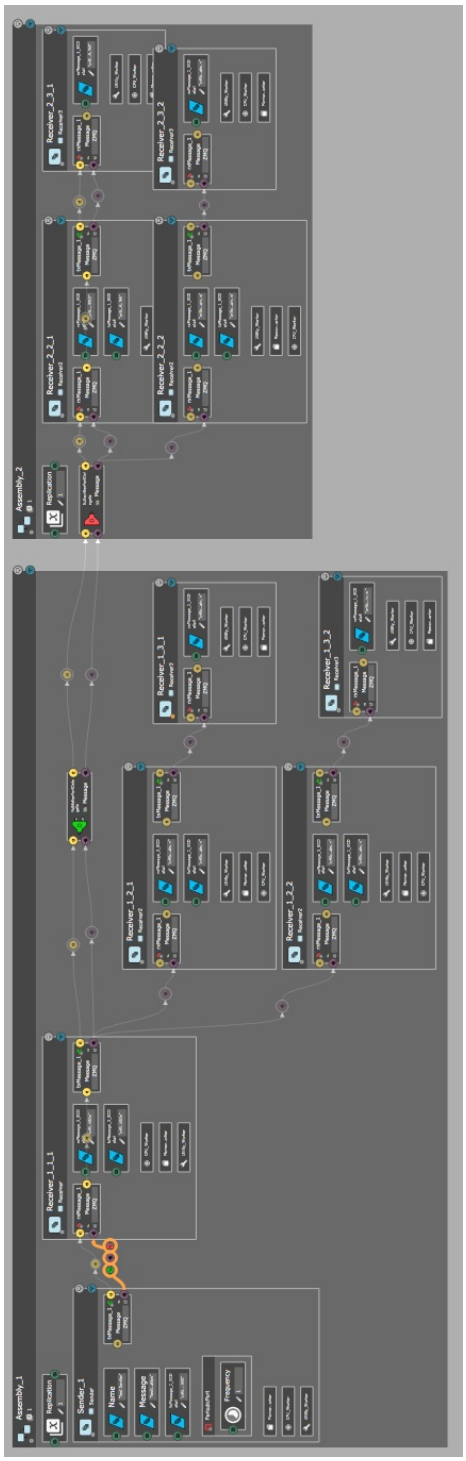

Figure 5.15: Critical Chain Trace

Included in the end-to-end latency calculations is the time it takes for software component workloads to be processed by the CPU, the time needed to transmit data via the network interface and transmission times through the network.

Before any time calculations can occur, utilisation profiles have to be constructed for each node within the computing environment that are available for deployment. This occurs as the Objective Score calculation includes the effects of resource utilisation and resultant delays on processing and transmission times. Resource utilisation profiling considers all of the software components to be deployed, not just those associated with the critical design thread.

As a result of needing utilisation mapping details for the Objective Scores, each new deployment profile for each search requires a complete new mapping of the software component to the nodes, alongside the resultant resource utilisation mapping. Once established, the total resource demands are translated into utilisation percentages for the CPU processor and network interface for each node available for deployment. When required later in the Objective algorithm, these values are then used in combination with the resource utilisation performance curve to calculate resource utilisation timing delay multiplication factors.

The execution flow for the calculation of the critical thread end-to-end latency can be seen in Figure 5.16, while Figure 5.17 shows a further breakdown of the element processing for each progress step along the CriticalChain.

Details for the different parts of this execution flow can also be found in Appendix C.2.

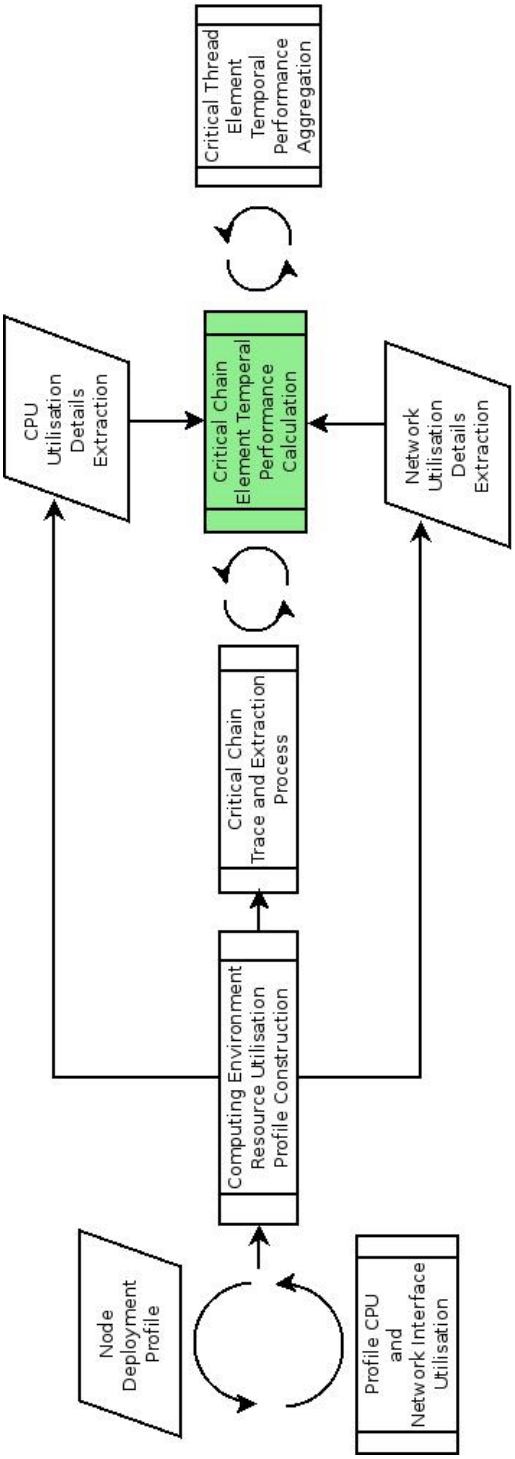

Figure 5.16: Critical Thread Temporal Performance Calculation Flow

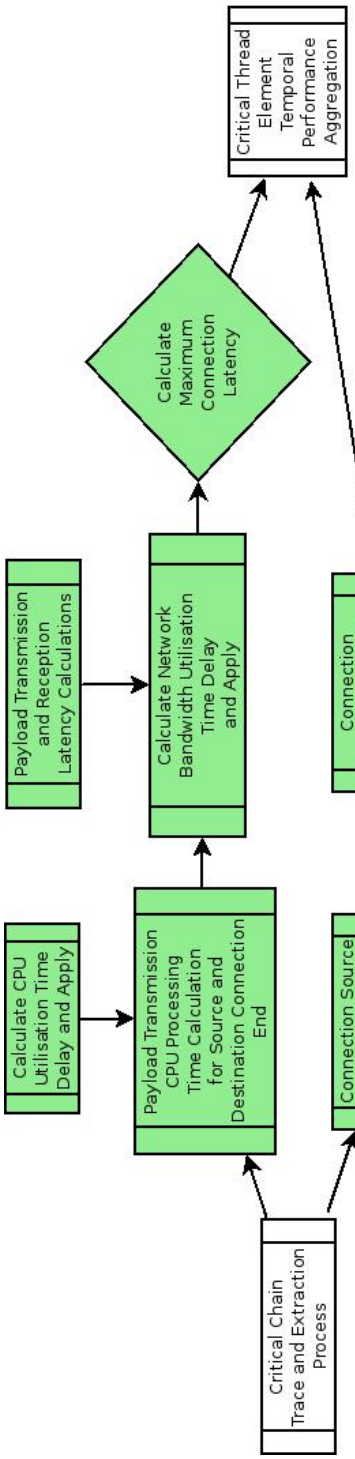

Figure 5.17: Critical Chain Element Temporal Performance Calculation Execution Flow

The second component of the Objective Score calculation is how well the deployment profile satisfies the identified non-functional requirements and constraints defined within the System Optimisation model. Through testing how well satisfaction is accomplished against the non-functional requirements or constraints, scores are produced and added to the core CriticalChain end-to-end latency performance score calculated previously. In some cases, these satisfaction scores may result in a negative score and the overall Objective Score being reduced at that point in the calculation process.

A formal definition of the objective function is as follows:

Let d be the software deployment options. The total Objective Score for that particular deployment d is then:

$$\Theta(d) = \Theta(CC) + \sum \Delta C_{ut} + \sum \Delta M_{ut} + \sum \Delta I_{lat} + \sum \Delta S_{lat} + \sum M_c + \sum M_a + \sum NC_s \tag{5.1}$$

where

$\Theta(CC)$ is the core critical thread end-to-end latency and CC is a component within that critical thread,

ΔC_{ut} is the CPU utilisation satisfaction score,

ΔM_{ut} is the memory utilisation satisfaction score,

ΔI_{Lat} is the Interface maximum latency satisfaction score,

ΔS_{Lat} is the String maximum latency satisfaction score,

S_{Man} is the software component mandate deployment satisfaction score,

A_{Man} is the assembly mandate deployment satisfaction score,

S_{Colo} is the software component no co-location mandate deployment satisfaction score.

and:

$$\Theta(CC) = \sum P_{kp}U_c + \sum N_{kp}U_c + \sum Tx_{kp}U_n \tag{5.2}$$

where

P_{kp} is the software processing latency,

N_{kp} is the network message processing latency,

Tx_{kp} is the message network transmission latency,

U_c is the CPU utilisation on node $p, p = \overline{1,m}$,

U_n is the network interface utilisation on node $p, p = \overline{1,m}$,

k is the number of components within the critical thread $k, k = \overline{1,n}$ on node p.

In the case of computing resource consumption constraints and latency requirements, the satisfaction scores are established via the use of a performance curve, whereby a growth multiplier is chosen and applied to each satisfaction score. These growth multipliers are proportional to the proximity of the predicted performance result compared with the required non-functional constraint or requirement modelled in the System Optimisation model.

The non-functional constraint and requirement Objective Score calculations are for the following:

- CPU spare capacity,

- Memory spare capacity,
- Connection maximum latency,
- String maximum latency,
- Component mandate deployment,
- Assembly mandate deployment,
- Component no-colocation mandate deployment.

The CPU and memory spare capacity Objective Scores calculations firstly require the node mapping process to determine the overall computing resource consumption for each of the computing nodes available for deployment. These resulting overall usage amounts are then compared with the non-functional constraints defined for CPU and Memory spare capacity requirements found in the System Optimisation model. The difference between the predicted and required non-functional constraints is then used to determine the selection of the growth multiplier function to be applied to the raw objective score.

For both CPU and memory this selection is based on two threshold percentage levels, as well as the 100% resource utilisation point. Figure 5.18 shows a spare utilisation threshold of 50%, an 80% hard threshold point and the 100% utilisation point. In the case of the hard resource utilisation level, this is a hard-coded level and is based on the common practice of having a utilisation ceiling of 80%, but it can be changed to any desired level.

As can be seen from Figure 5.18, the choice of growth multiplier (and subsequent scaled Objective Score) is such that optimisation scores are scaled up (remembering the lower the Objective Score the more optimal the search is) as

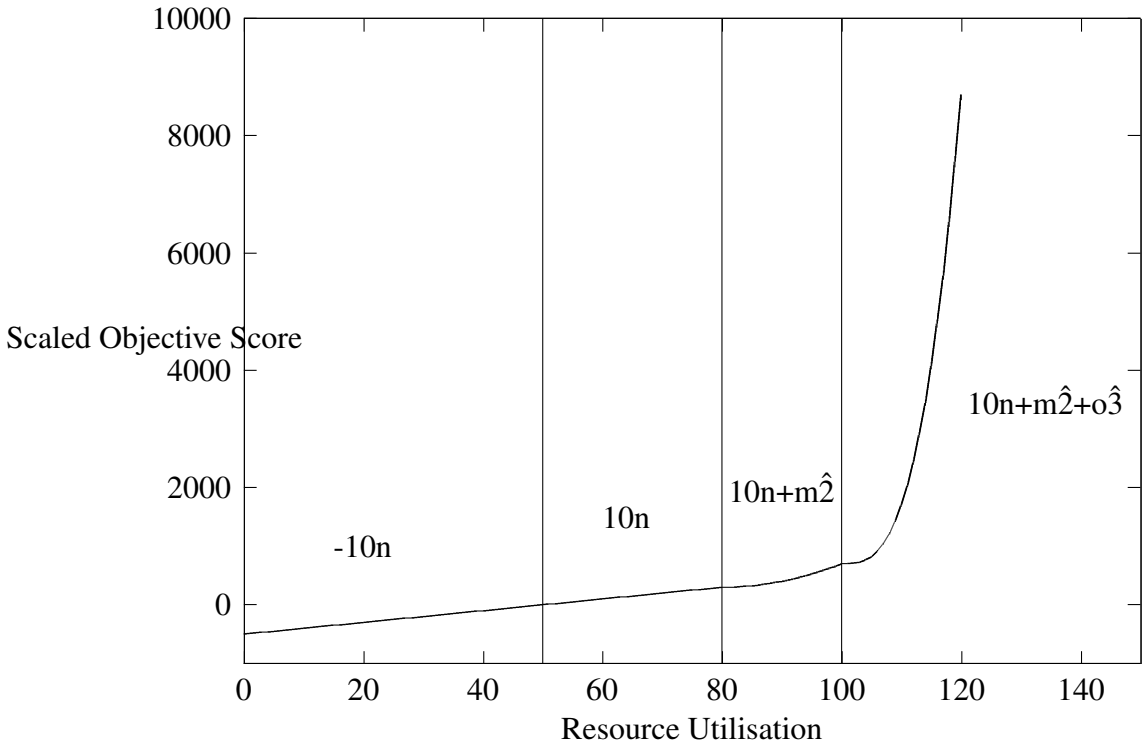

Figure 5.18: Example Resource Utilisation vs Scaled Objective Score

the predicted utilisation passes beyond the spare capacity. This scaling up of the Objective Score increases rapidly beyond the hardware threshold level, and even more rapidly as the predicted utilisation goes beyond the 100% threshold. On the other hand, if the predicted utilisation is less than the desired spare capacity, the Objective Score is scaled down with a negative growth. This then produces a reduction to the overall Objective Score, which indicates an ideal, optimal solution.

The four sections of resource utilisation, associating attributes and growth multiplier functions are as follows:

$$Current\ Resource\ Utilisation -> \mu$$

$$Resource\ Hard\ Threshold\ Utilisation -> \mu_t$$

$$Spare\ Capacity\ Level -> r_s$$

and:

$$Growth\ Function = \begin{cases} -10n, & 0 < \mu < r_s \\ 10n, & r_s < \mu < \mu_t \\ 10n + m^2, & \mu_t < \mu < 100 \\ 10n + m^2 + o^3, & \mu \geq 100 \end{cases} \tag{5.3}$$

where:

$$f(n) = \begin{cases} r_s - \mu, & 0 < \mu < r_s \\ \mu - r_s, & r_s < \mu < \mu_t \\ \mu_t - r_s, & \mu_t < \mu < 100 \\ \mu_t - r_s, & \mu \geq 100 \end{cases} \tag{5.4}$$

$$f(m) = \begin{cases} \mu - \mu_t, & \mu_t < \mu < 100 \\ 100 - \mu_t, & \mu \geq 100 \end{cases} \tag{5.5}$$

$$f(0) = \begin{cases} \mu - 100, & \mu \geq 100 \end{cases} \tag{5.6}$$

The connection and string (remembering this is a segment of the critical thread) maximum latency Objective Score calculations make use of the timing performances calculated during the critical thread Objective Score calculations. These calculations are then compared with the temporal non-functional requirements defined in the System Optimisation model.

Once again, an Objective Score growth functions are used to scale the raw Objective Score, and the choice of growth function is dependent on how close the predicted performance is to the modelled performance threshold. In the case of both the connection maximum latency and string maximum latency being used, only a single threshold is utilised, and two growth functions are called upon.

In each case, the threshold is related to the maximum latency that is acceptable

for a particular deployment, where this threshold is calculated as a comparison percentage of the predicted maximum latency time compared with the desired maximum latency time. As with the resource utilisation Objective Score scaling, predicted performances that exceed this threshold are rapidly scaled up, while performances below the threshold are scaled down with negative growth and result in a reduction of the overall objective score.

The sections for maximum latency and associated growth functions are:

$$Current\ Comparison\ Percentage -> \rho$$

and:

$$Growth\ Function = \begin{cases} -10n, & \rho \leq 100 \\ n^2, & \rho > 100 \end{cases} \tag{5.7}$$

where:

$$f(n) = \begin{cases} 100 - \rho, & \rho \leq 100 \\ \rho - 100, & \rho > 100 \end{cases} \tag{5.8}$$

Additional scaling can also be applied for the constraint Objective Scores through the use of the weightings modelled for each. For the latency Objective Score calculations, a single, defined weight is applied to the overall Objective Score calculated for connection and string constraints. However, the resource utilisation Objective Scores are calculated on a per node basis, and the weight is applied for

that particular node alone.

The modelled deployment constraints serve the Objective Score calculation in two ways. The first is whether the current search solution is to be included or not, where inclusion means all the constraints are satisfied. The other is when the search wants to consider search solutions that do not satisfy all the mandated deployment constraints. In this scenario, a direct addition of satisfied and non-satisfied constraints are determined and weights are applied to each.

Figure 5.19 shows the execution flow for the Overall Objective Score calculation.

Final generation of population

The output of the hybrid optimisation process leads to a final set of optimal deployment profiles, based on the Objective Score of the final search solution population, which occurs after the creation of a defined number of generations. This final population is then used within the graphML file creation process to construct the new MEDEA graphML files for the original system design model. However, these new files now hold the node deployment details to dictate where each of the software components are to be deployed.

This set of newly-created graphML files are then fed back into the original MEDEA execution flow to enable experimentation, followed by analysis of the performance gains from those optimised deployment profiles.

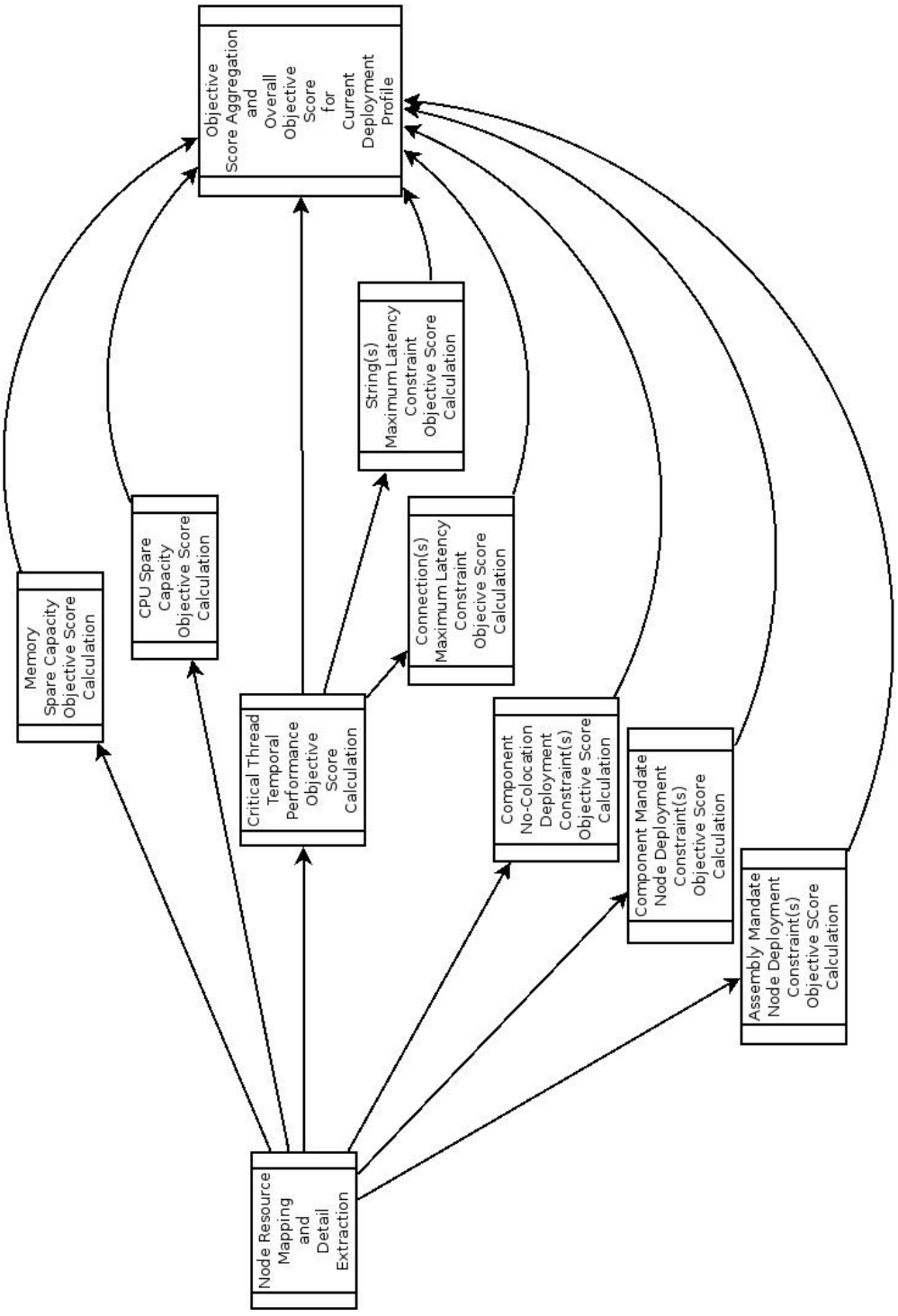

Figure 5.19: Overall Objective Score Calculations

6. Verification and Calibration

In the previous chapter, a new approach was introduced for identifying software deployment solutions that deliver optimal performance for known non-functional design aspects. It introduced new modelling elements and methods integrated into the MEDEA modelling environment, information translation mechanisms, and run-time instrumentation and analysis frameworks. It also introduced the core elements of the approach, the Evolutionary Search and Objective Function algorithms.

This chapter will detail new frameworks, methods and results to demonstrate the approach introduced within this thesis operates as expected. It will verify the Objective Function and Evolutionary Search algorithms are performing as expected, and demonstrate the ability of the introduced capability to construct system designs that deliver optimal performance for the known non-functional design aspects. It also details calibration methods to improve algorithm performance, along with results to demonstrate the improvement.

6.1 Objective Function Algorithm Verification and Calibration Requirement

As described in the previous chapter, at the core of the Objective Function is a calculation of the predicted temporal performance of the critical thread of interest within the system design. Furthermore, the time component calculations (for workloads, component interfaces and middleware transmission) that make up this core objective score are used for internal constraint objective score calculations, which then add to the overall objective score calculation.

With time prediction a major element of the Objective Function algorithm and a significant influence on the overall objective score, we focus on how well the prediction of time compares with measured time. This is then followed by determining the accuracy of the time representation to ensure a correct influence from temporal performance constraints on the overall objective score.

To determine these factors, we used a specialised test model where the overall objective score is based only on the critical thread temporal performance. Then, by deploying and executing this test model, measurements and analysis were conducted to determine how close the predicted critical thread time is to the measured times.

In addition to understanding the performance and accuracy of the prediction of the critical thread times, a second test model was deployed to allow for a direct comparison of the raw temporal performance predictions (without the final Objective Function final adjustment added) with the measured times. These tests

then provided insight into how well the prediction algorithms account for time, and whether there is a need for an adjustment factor to improve the time representation and prediction performance of the algorithms.

While efforts have been made to provide an improved level of fidelity for the prediction of time, the models used in this approach are not a 100% representation of time. Subsequently, certain population members may be added to generations in error and result in optimised deployment profiles being identified in error, or possibly not identified at all. Follow-on research efforts would consider and evaluate improved models for time representation that would improve the fidelity and performance of the Objection Function and reduce the possibility of non-optimal solutions being produced.

Verification Procedure

To test the performance of the optimisation search and its ability to represent time, a generic baseline testing model was developed. This baseline model consisted of four component definitions: a sender component (Figure 6.1), which initiates the transmission of messages through the system, and three receiver components. In this test case, the transmission of the message occurred at a constant rate of once per second.

The receiver components have three different forms: to represent a communication conduit for reception and transmission with associated workloads (Figure 6.2), a switching of workloads with certain logic on reception and subsequent transmission (Figure 6.3), and to simply receive the message and have its associated

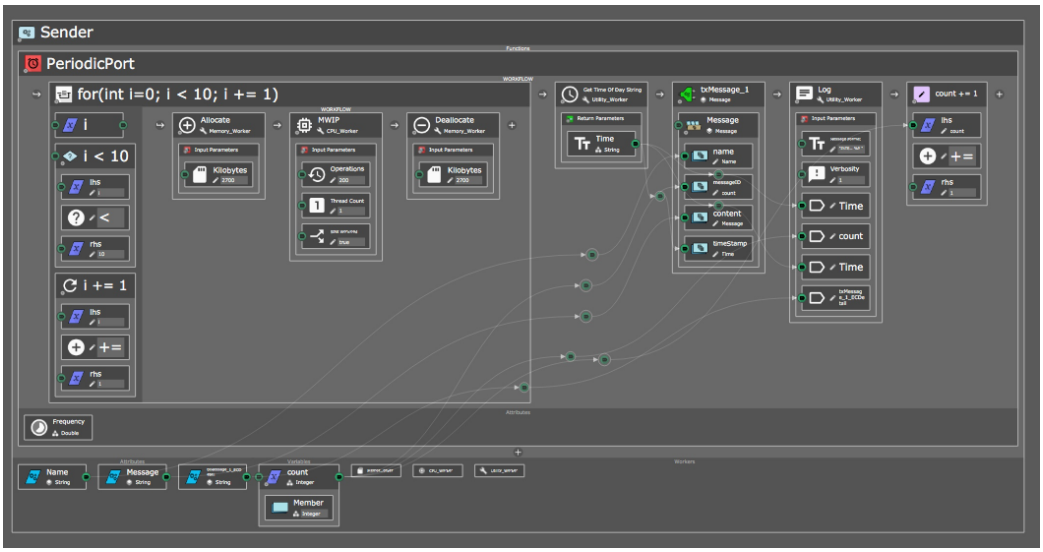

Figure 6.1: Test Model Sender Component Definition

workload executed (Figure 6.4).

Figure 6.2: Test Model Receiver Component Definition: transmission conduit

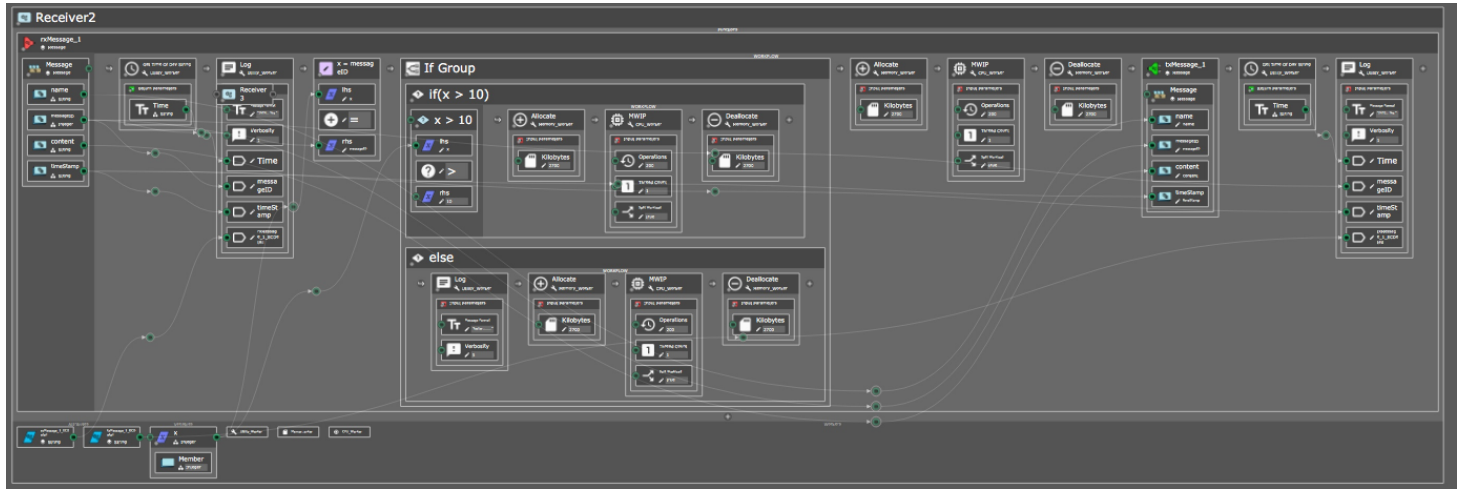

Figure 6.3: Test Model Receiver Component Definition: switching workload

Figure 6.4: Test Model Receiver Component Definition: receive only

Furthermore, the workloads defined within each component, 15 in total across all component instances, were the same size and based on a number of factors derived from the resource specifications of the computing environment available for the test exercise. These factors are detailed below. Tables 6.1 shows the results of these factors being:

- equal in size, where the size is based around the node with the largest resources available,
- the Objective Function algorithm hard CPU threshold level (80% in this case) of the computing node with the largest resource,
- an even distribution of workload across the workers within the test model,
- applying a multiplication (50% increase) to prevent all the components from being deployed to one node.

Factor	CPU Workload	Memory Footprint
the largest node resource	2579 MWIPS	3364MB
the 80% threshold level	2000 MWIPS	2700MB
each worker within the model	133 MWIPS	180MB
50% increase factor applied	200 MWIPS	270MB

Table 6.1: Results of the application of the workload factors

Therefore, each workload defined within the test baseline model has a CPU worker consuming 200 MWIPS and a memory worker consuming 270MB. In addition to the consistent sizing of CPU and the memory consumption deployed, all the messages transmitted were the same size and maintained the same transmission frequency throughout the test model.

The reasoning behind this consistent and constant configuration of resources was to ensure that the changes in resource utilisation put on each available node were dependent on the number of workloads added to that node, and not related to a particular workload characteristics added to the node.

From the four component definitions, 10 component instances (Figure 6.5) were created to form the test model and a SoS representation suitable for search algorithm performance verification testing.

Figure 6.5: The Testing and Tuning System Model

Verification Framework

To support the verification effort, additional measuring mechanisms were developed and augmented with the existing MEDEA logging infrastructure. This included the injection of new messages within the message header of the existing MEDEA infrastructure. As a result, additional details on the relationships and performances of the elements that made up the modelled system design thread of interest were captured in readiness for analysis.

All the new information created was also captured in the SQL database created out of the MEDEA execution logging framework, and a standalone text file.

The additional messages and attributes captured within the MEDEA SQL database (as a series of text strings) are are as follows:

- **Message header: 'INTER'**
    - The INTER text string indicates that this logging event has captured a flow of the critical thread that is between the software components, utilising the middleware and network connection available within the MEDEA run-time computing environment. It is the flow from the output port of a software component to the input port of another software component and both are identified as part of the critical path of the system design model.
- **Message header: 'INTRA'**
    - The INTRA text string indicates that this logging event has captured the flow of the critical thread that is within a software component. This could be via an internal periodic event or the reception of a message

via the input port of the software component. It indicates the flow is about to execute workloads associated with the input and output ports, as well as any associated logic found within that software component.

- *Message header:' ECDetail'*
  - The ECDetail text string captures the modelled information on a segment of the identified critical path. This includes details on the system model elements (graphML id information) that make up the particular segment, non-functional performance requirements and segment positioning detail.

Further details on the message set can be found in Appendix D.1.

To make use of the additional information added to the MEDEA SQL database, as well as the standalone text file, an information extraction, association and aggregation framework was created. The processing steps within this framework consisted of numerous extractions and translations, associations and aggregation steps to develop the required new data sets. Using these data sets analysis and confirmation of correct performance for predicted latency and resource consumption occurred, as well as search algorithm performance and the correctness of solutions produced for deployment.

Figure 6.6 shows the framework execution flow with information processing stages.

6.2 Objective Algorithm Performance Verification

As detailed above, the first test model created was only concerned with the temporal aspect of the system performance and did not introduce any non-functional constraints into the optimisation search. This, therefore, enabled exploration as to how well the predicted temporal performance compared with the measurements of the temporal results of the critical thread.

This validation approach had two steps. The first step compared the complete population set and their objective scores (which was also the predicted temporal performance) with the measured temporal performance for each member of the population set.

The second step of the validation process determined how the optimisation solutions drive consumption of computing resources on each computing node, and whether observed levels of consumption are following the desired characteristics and impacting non-functional performance accordingly.

6.2.1 Validation Experimentation of Initial Conditions

The initial conditions for the validation experimentation were a solution population of ten for each generation, the creation of solutions for twenty generations and each member of a particular population for each generation to execute its particular deployment profile for sixty seconds within the MEDEA run-time environment (NB: 58 messages are only used for the analysis process to account for anomalies

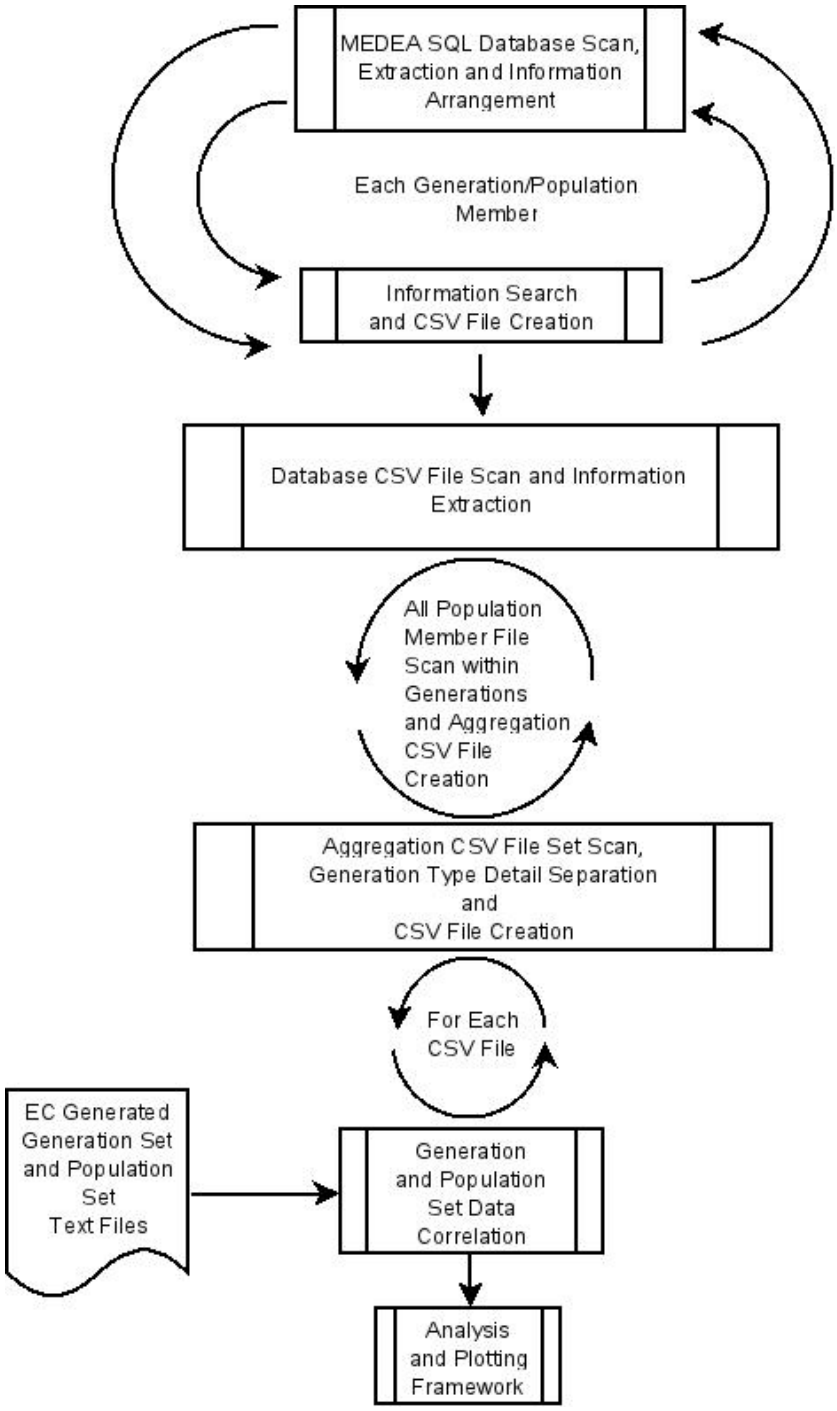

Figure 6.6: Information Processing Flow

and loss of data during the destruction of each execution within the Jenkins environment). Furthermore, during this validation experimentation phase no changes were made to the test computing environment available for deployment, other than exploring different network bandwidths.

The results of these individual runs were then downloaded from the MEDEA run-time environment in their SQL database to form a repository in readiness for input into the information processing flow depicted in Figure 6.6.

6.2.2 Validation Results

As part of establishing the goodness of the measured data sets for each population member, we report the standard deviation and confidence interval calculations. As indicated above, these calculations only consider 58 measurement samples of each population member, while the confidence level calculations consider a 95% confidence level Dekking et al., 2006.

Figure 6.7 shows the sample mean and standard deviation for each population member of the overall population, while Figure 6.8 shows the sample mean for each population member's measured data set, as well as an indication of how well the data set sits is within the 95% confidence level of the population mean for all possible measurements. Both graphs in these test examples show a large portion of the population data sets are of good quality with relatively low standard deviation across the sets and a high level of confidence. The sets are within the 95% level of the measurement population mean. Furthermore, while poor population data sets are seen within these test scenarios, it is reasonable to assume that because of the

low latencies being measured the effect of jitter (or packet transmission periodicity deviation) may well be a factor in these lower quality data sets.

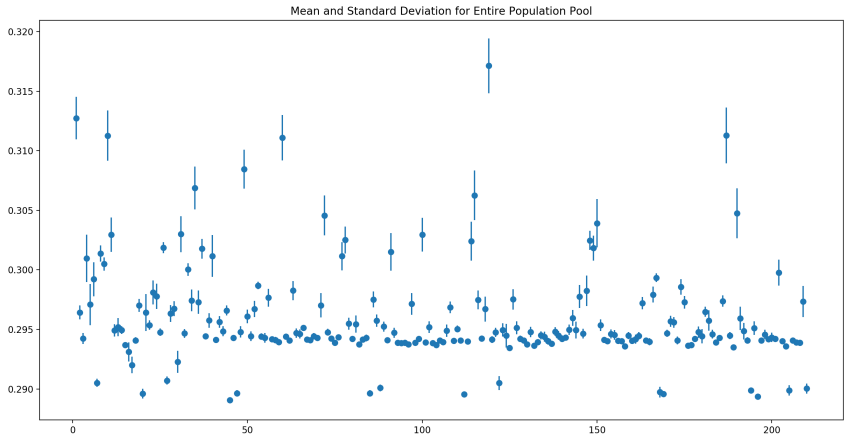

Figure 6.7: Standard Deviation of Measured Data Sets for the Calibration Process

After the quality of the population data sets was established, the next validation step was to understand how well the predicted performance matches up with the measured performance for each population member of the overall population. This was achieved with two independent test steps.

Using an established method of validating prediction algorithms (Akdemir and Jannink, 2015, Santtila et al., 2013, Pratt et al., 2004, Petersen et al., 2009, Wong et al., 2012 and D'Ambros, Lanza, and Robbes, 2010), the first calculation was a Pearson Correlation Coefficient (using the standard Python scipy stats library [1]), along with the development of a scatter and distribution plot for the predicted latencies versus the mean measured latencies.

[1] https://docs.scipy.org/doc/scipy/reference/stats.html

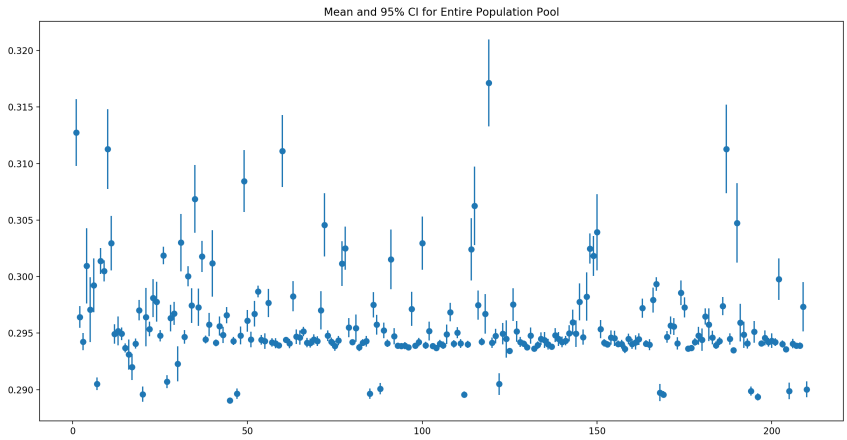

Figure 6.8: 95% Confidence Interval of Measured Data Sets for the Calibration Process

The second of these calculations was to determine the polynomial fit for each of the predicted (remembering the prediction algorithm has been configured to only predict the critical thread temporal performance) and mean measured data sets for the latencies. This then enabled a comparison of the curves for each, and insight into the performance of the predictions based on the alignment of the curves.

Figure 6.9 shows the results from one test run to confirm how well the predicted results aligned with the measured latency results. This plot portrays the results as scatter and distribution plots, as well as a Pearson Correlation Coefficient calculation. In this case, we see the Pearson Correlation Coefficient being r=0.632, indicating a strong relationship [2] between the predicted and measured data sets, as well as a reasonably low p value indicating that 99.97% of the time the relationship

[2]The use of strong is based on the reference Akoglu, 2018

is at this strong level.

Furthermore, we observe the distribution plots for both have a reasonable alignment, in phase, and both exhibiting a convergence towards a minimum. While the distribution spread at the minimum is not ideal and does not peak at the smallest latency, the observed behaviour indicates the predictive nature of the objective algorithm is trying to minimise latency. Future research efforts aiming to make improvements in the models used for time representation and temporal performance would lead to further improvements in the convergent behaviours observed here[3].

From these plots it can be seen that the tests have a moderate to strong relationship[4] with the Pearson Correlation coefficients ranging from r=0.345 to r=0.513. Furthermore, the results have a high levels of p values (the worst being a 99.96% of the time the r value will be maintained). Overall, there is a strong indication of good predictive performance for the optimisation algorithm to seek deployment profiles with minimal latencies for message transmissions across that critical thread of interest.

Figure 6.10 shows a set of results from the second calculation method, where a polynomial fit for each data set occurs, followed by a correlation between the curves across the entire population. Appendix D.3 shows the additional results of the polynomial calculations.

As with the Pearson Coefficient tests, the polynomial fit tests occurred across different network bandwidth configurations. Furthermore, the plots are presented

[3] Appendix D.2 shows the Pearson Correlation coefficients and corresponding distribution for other validation test executions across differing network bandwidth configurations.

[4] The use of moderate to strong is based on the reference Akoglu, 2018

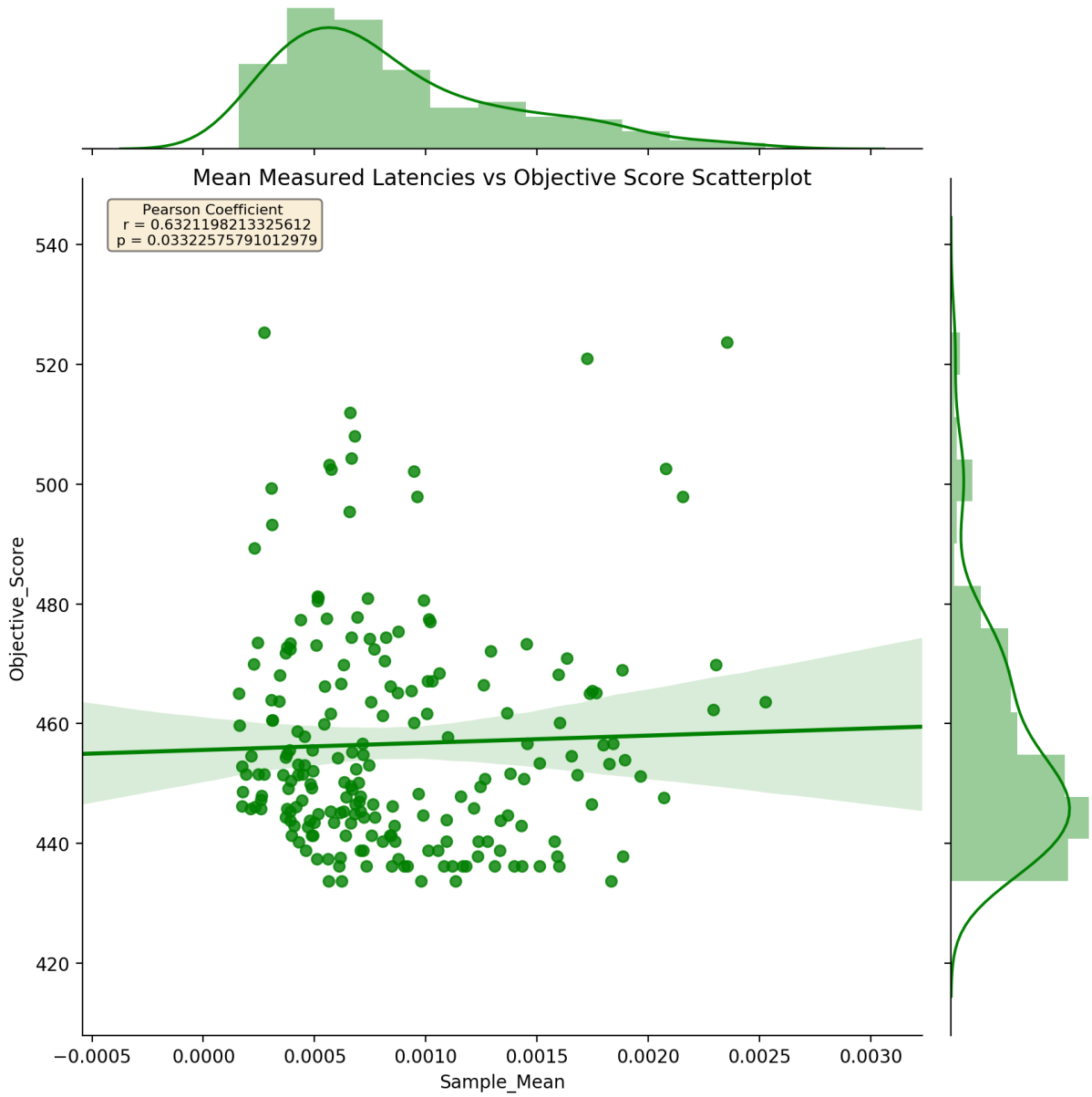

Figure 6.9: Scatter Curve and Pearson Correlation Coefficient for the Mean Sample Versus the Predicted Latency with r = 0.632

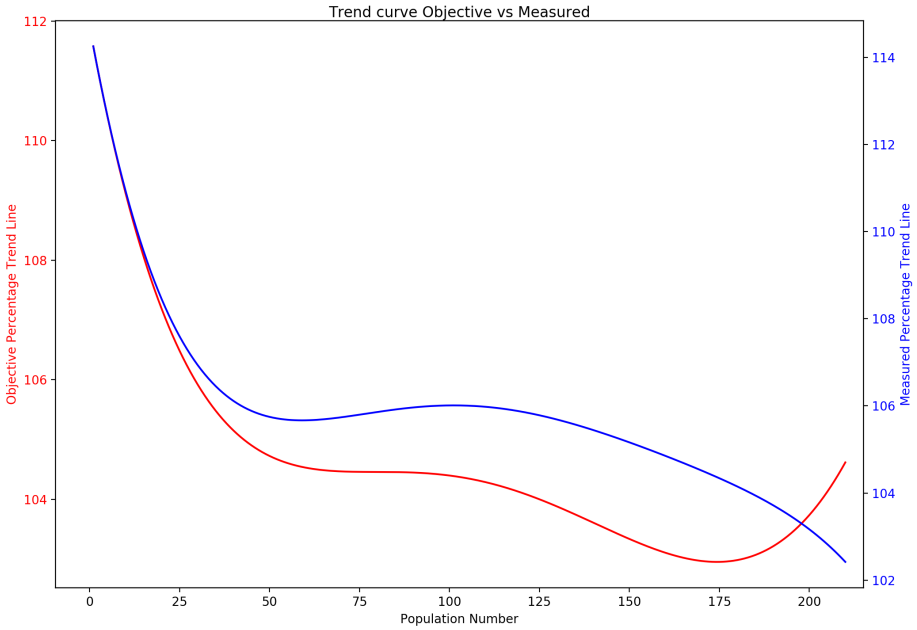

Figure 6.10: Normalised Percentage Curve for the Average Latency versus the Predicted Latency

as a value of the percentage related to a minimum value of each data set.

As can been seen in the test results, both the predicted and measured data sets have a reasonable alignment with each other in both magnitude and phase. While we observe some size differences in magnitude for the test results, the largest difference is less than 5%. These results further reinforces the observation made from the first calculation process that the predictive performance of the optimisation algorithm is at a reasonable level, and more importantly demonstrates that the algorithm is exhibiting the desired outcome of minimising latency through its search capability.

As detailed in the previous section, the optimisation algorithm considers resource utilisation levels. This not only focuses on the impact to the overall computing node resource utilisation, but also the impact of resource utilisation leading to timing delays and resulting temporal performance. Furthermore, due to network workload being constant and CPU workload being the only variable, this only extends to CPU utilisation and its impact on temporal performance.

In the first test model deployment scenario, when no definition of constraints is present to influence the search, the deployment of software components onto computing nodes and consumption of their resources was based on ensuring the associated time delays are as low as possible. It would therefore be reasonable to expect (in general) that utilisation of CPU processing resources across the node should be minimised as the number of generations increases.

Figure 6.11 and Figure 6.12 show two sets of test results for utilisation for CPU processing resources across the four available computing nodes for software

deployment. The first set of results shows an envelope for maximum CPU utilisation for each generation across the generations, while the second set shows the maximum CPU utilisation for each population member within the generation, across the generation.

In line with the expected behaviour, we can see that the resources for CPU do in general decrease in utilisation across the generations. From the maximum CPU utilisation across generation graphs (Figure 6.11), we see the decrease for each node and a leveling out to an approximate similar CPU utilisation. While the maximum utilisation for each population across generations (Figure 6.12) shows an initial scattering within generations and then the convergence with the increase in the generation count. Both are expected results and in alignment with the prediction of requiring equally dispersed workloads to minimise time delay and impact on the overall temporal performance.

Based on these results, it can be further concluded that the optimisation algorithm is behaving as desired, achieving reasonable levels of fidelity and accuracy for representation and prediction of time performances. It also shows the algorithm's ability to change software deployment options in response to its solution exploration.

6.2.3 Time Comparison Trend and Size Calculations

Following the confirmation of adequate levels of accuracy and desired prediction performances by the optimisation algorithm, the final step of the testing process was to conduct a series of comparison measurements to determine how accurate

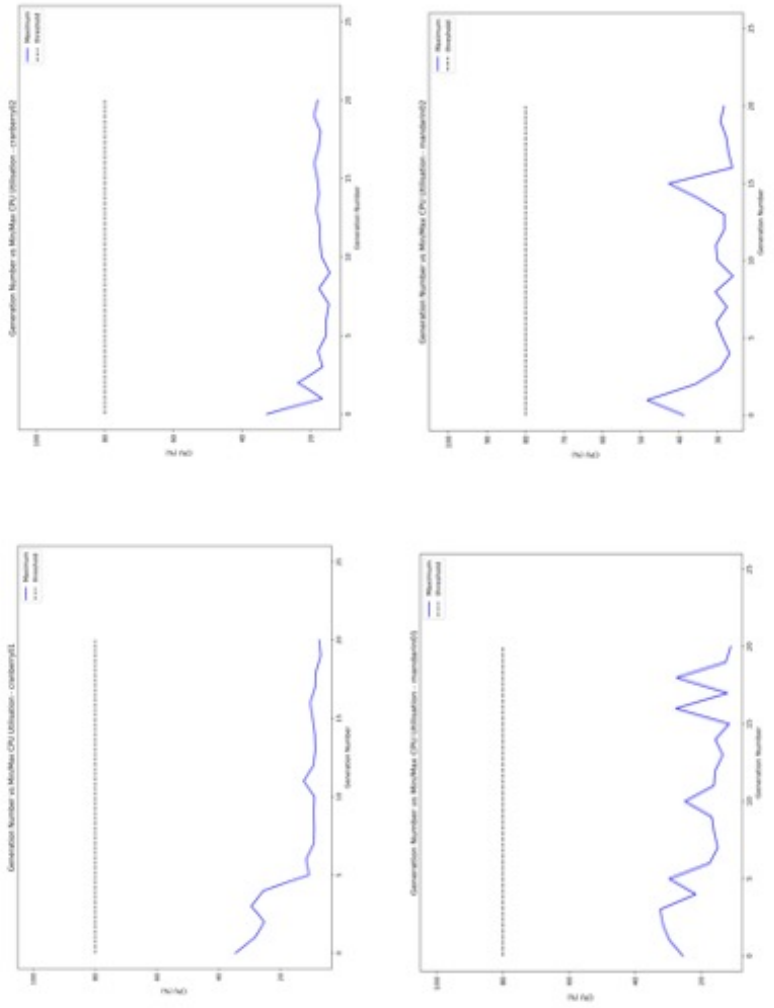

Figure 6.11: CPU Node 1 to 4: Maximum Utilisation Results Per Generation

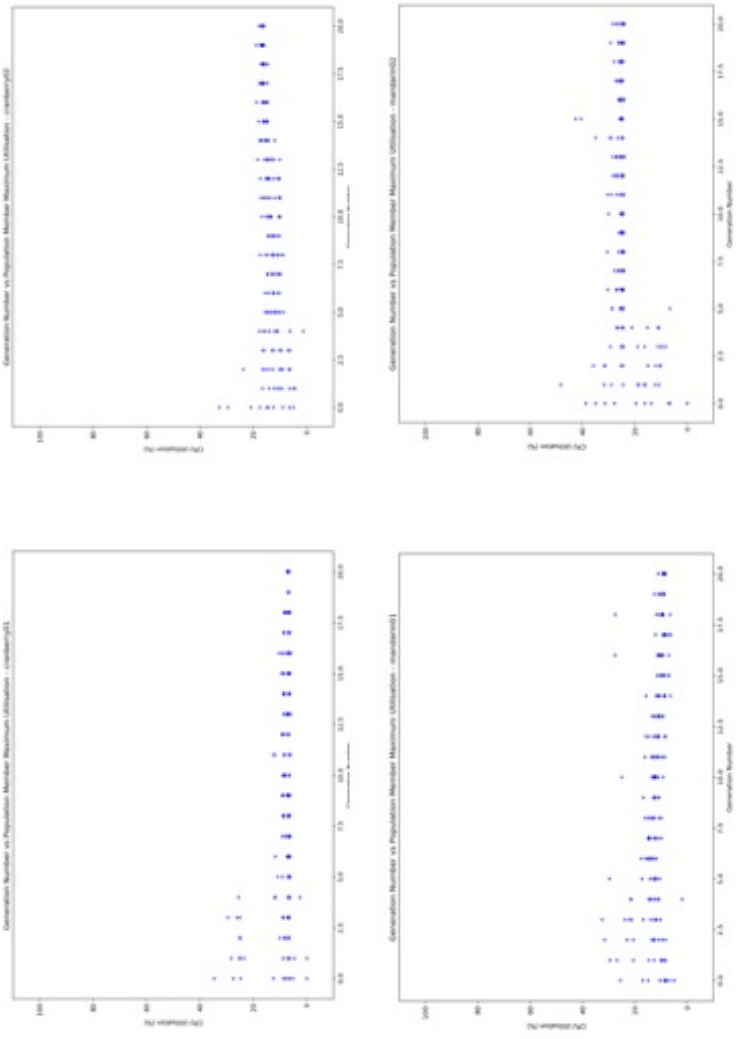

Figure 6.12: CPU Node 1 to 4: Maximum Utilisation Results Per Population Member Per Generation

the representation of time is across the numerous test runs. The results from these tests then determined the trend and size of the difference between the predicted time and the measured time across a large set of test results. These results were then used in combination with the temporal requirement performance prediction comparison calculation (Chapter 6.4) to establish adjustment factors to be added to the Objective Function.

The calibration approach used was a straight forward comparison analysis of the predicted latencies versus the measured latencies, followed by capturing the deltas. The deltas were then averaged over the 58 measured samples versus the predicted latency and then averaged across the test sets.

Figure 6.13 shows a scatter plot of the results of a comparison of the predicted versus measured latencies. In this case, the plot is in milliseconds and shows the differences across the entire population, along with the average and standard deviation results. As can be seen from this plot, as well as other test and analysis runs, the performance of the latency prediction is constantly higher than what is measured.

Table 6.2 shows the results of averaging across numerous test runs, which were also conducted with differing network bandwidth configuration. From this table it can be concluded that the predictions of time performance are always greater than the measured performance and the size of the delta is reasonably consistent, sitting around the 33% point.

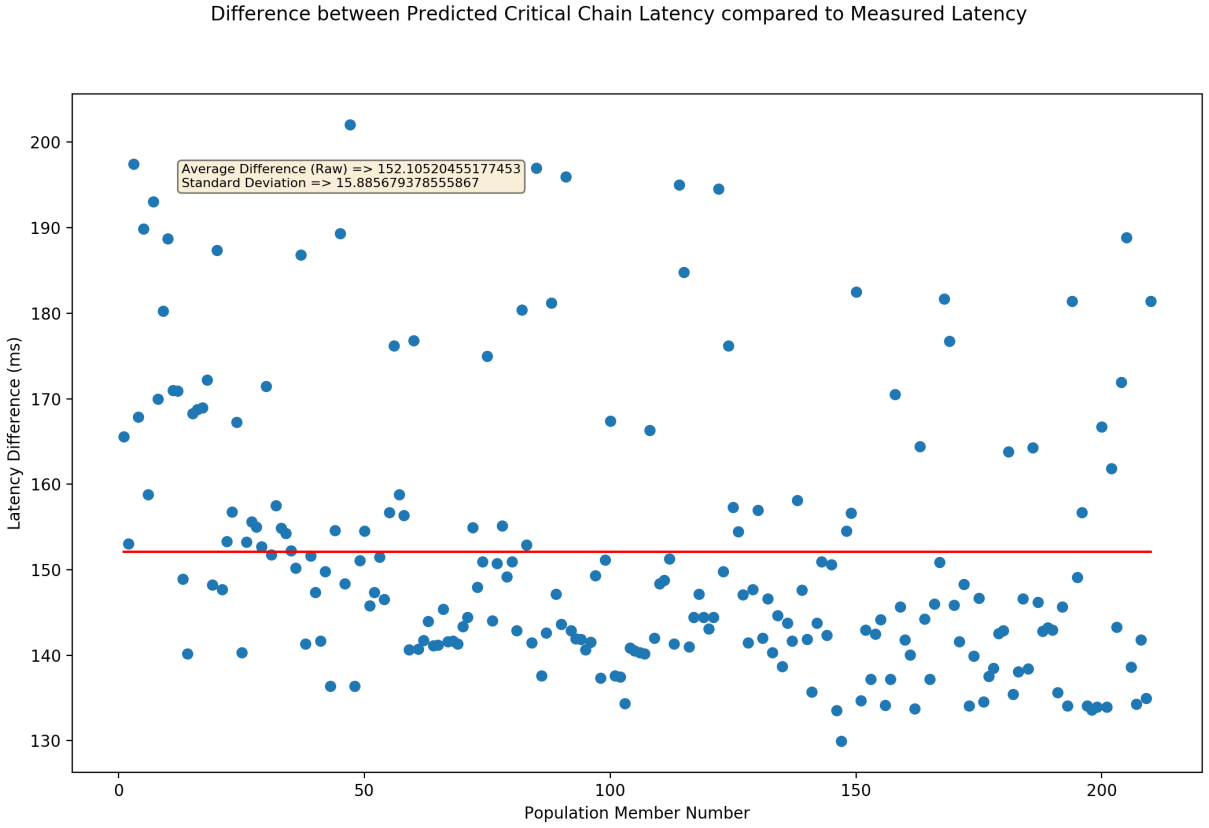

Figure 6.13: Scatter Plot of Predicted and Measured Latency

Set	$\overline{X}$ (ms)	σ (ms)	$\overline{X}$ (%)	σ(%)
1	158.315	19.406	34.563	2.774
2	152.105	15.886	33.874	2.246
3	148.844	18.208	33.128	2.773
4	148.841	18.286	33.125	2.790
5	157.318	19.710	34.342	2.845
6	148.989	18.232	33.158	2.769
7	148.841	18.286	33.125	2.790
8	151.272	16.508	33.684	2.367
9	151.462	15.765	33.731	2.228
10	158.634	19.574	34.741	2.651
11	154.735	20.969	33.876	3.077
12	156.593	19.377	34.514	2.605
13	156.577	19.332	34.510	2.593
Average	153.327	18.426	33.875	2.655

Table 6.2: Predicted Versus Measured Latency Comparison Results and Averages

6.3 Initial Generation Diversity and Search Algorithm Performance

In addition to testing the prediction performance of the Objective Function algorithm against measured results from execution of the deployment profile solutions, verification was performed to ensure the search algorithm was behaving and performing as expected. The areas of concern for this test were to ensure population diversity of the initial generation was adequate enough, and whether the nature of

the convergence and performance of the search algorithm was adequate.

To enable this analysis, a deployment profile number was added to the population details of each member of the population. This was generated by simply converting the deployment array of node id numbers into a binary representation, aggregating the binary numbers to produce a binary string, and then converting that string into an integer. In the test case, only four nodes were available for deployment, so the conversion process simply utilised two binary bits for the integer conversion.

In addition to proving a good level of diversity within the randomly created initial population, the deployment numbers were also tracked across the generations created to gain insight into the search algorithm performance. This should then produced results that show a good spread of deployment profile numbers at the beginning, followed by convergence onto optimised search solutions, or possibly a single optimised solution.

Figure 6.14 shows the results of an analysis of an optimisation search for a population of 100 across 40 generations. From this graph we can see the randomly created initial generation population (Generation Number 0) has a good level of diversity for the chosen deployment options from all the possible options available. Subsequent generations created from the initial population begin to show the convergence towards optimal solutions and, in fact, from about generation five the results show convergence in four particular areas. At about generation 15 we see convergence towards some maxima, followed by a drop off around generation 30 to a single maximum.

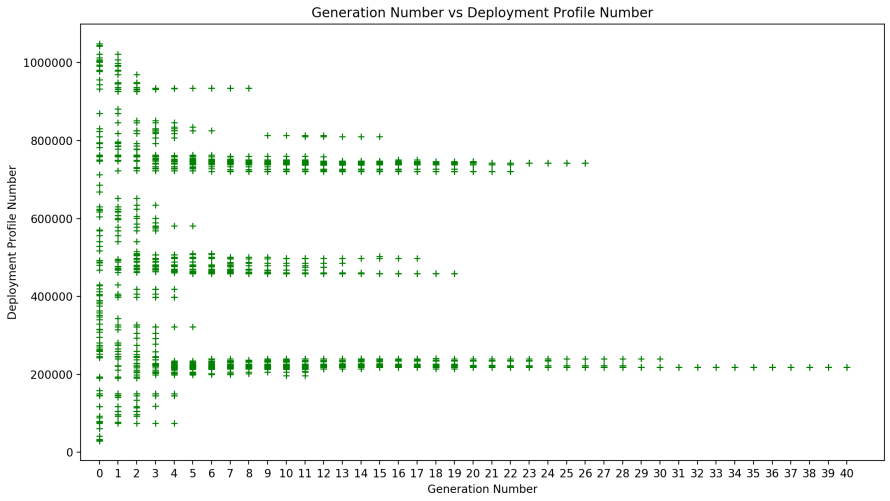

Figure 6.14: Results of the specialised populations and generations executed search

From these results, it can be concluded that, firstly, the random population creation for the initial generation is diverse in nature and creates a rich starting point for the optimisation search. Secondly, the search algorithm exhibits the expected converging behaviour, as well as performance in reaching these convergence points.

When comparing the results of the specialised population test run (Figure 6.14) with the results produced for the Objective Function test runs (Appendix D.4), observations can be made about the objective tests converging around different maximums. Furthermore, it can be seen that in some cases more than one optimised solution was identified.

In response to this, analysis was also conducted to confirm that when more than one optimised solution was identified, the identified solution's objective score was still being minimised.

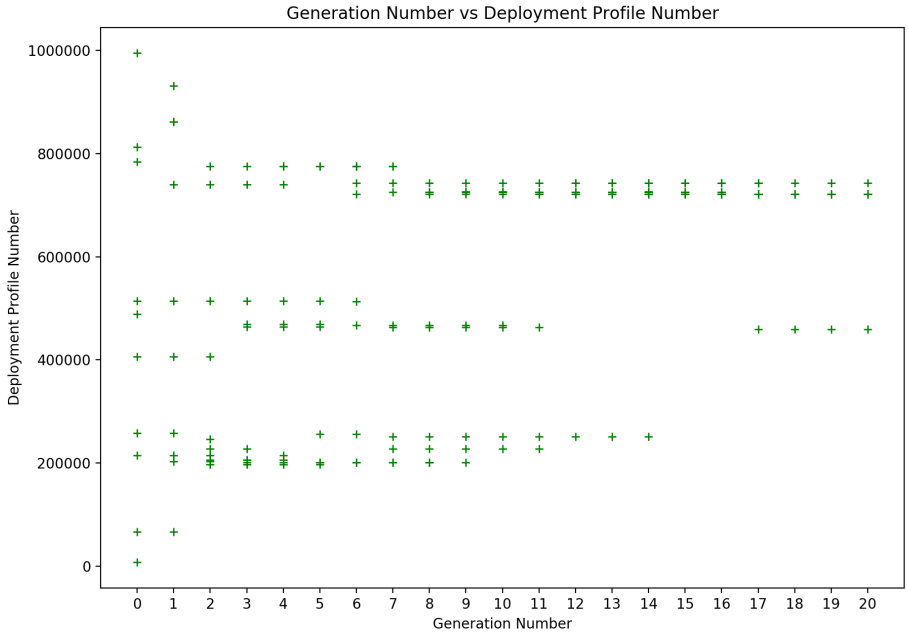

Figure 6.15: Deployment Profile Results #1

Figure 6.15, Figure 6.16, Figure 6.17 and Figure 6.18 shows some of the results from the deployment profile to objective score comparison analysis. From these plots we see that, while different and numerous maximums can be produced, all the associated objective scores across the generations consistently converge to a minimum, thereby confirming the correct behaviour and performance of the Objection Function optimisation capability.

In response to large model execution times within the MEDEA environment for each set of optimised deployment profiles (for example each set of 20 generations and population 40 requires a minimum of 24 hours), an investigation also occurred to identify a generation and population size for the subsequent Case

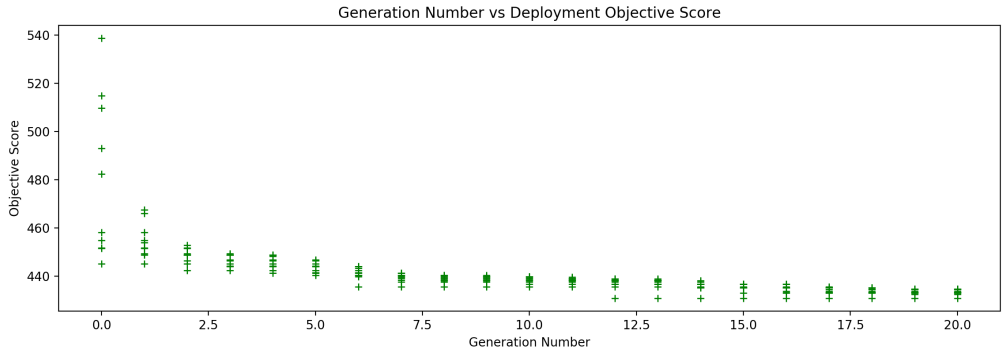

Figure 6.16: Objective Results #1

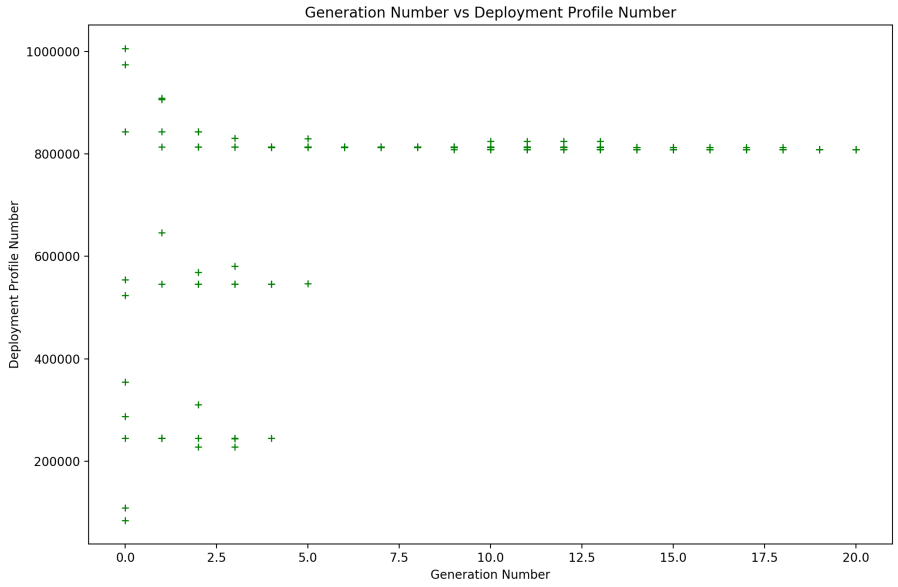

Figure 6.17: Deployment Profile Results #2

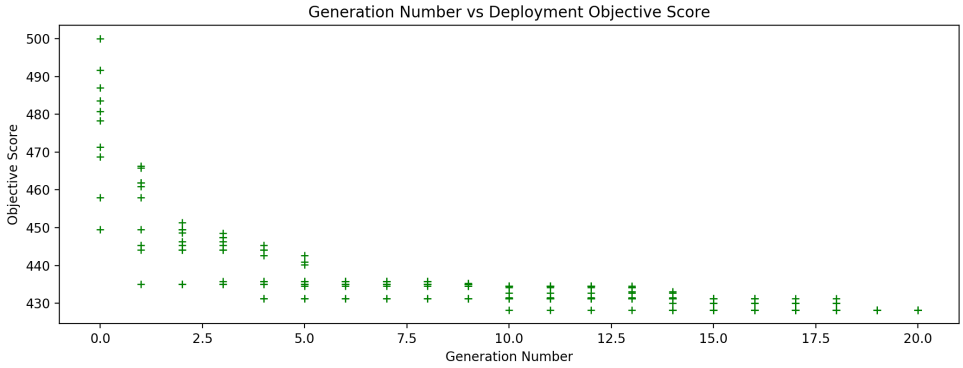

Figure 6.18: Objective Results #1

Study experimentation. The outcome of this investigation was to identify the size of the generation and population count that will ensure the required convergent performance, while not leading to non-viable execution times for experimentation.

Based on the results shown in Figure 6.14 where we observe convergence occurring at the 218425 deployment profile number from the 35th generation point, the investigation initially compared results for a generation size of 50 rather than 40. Following this, population sizes ranging from 60 to 100 were then investigated. Appendix D.5 has the results of one of the sets of generation and population permutations test runs.

From the plots for a population of 100 and a generation count of 40, we observed a convergence towards maximums across the plots, but there was a large variability for the converged results. For a population of 100 and a generation count of 50, we observe an improvement in consistency in the reaching the optimised outcomes and convergence towards the same maximum. As a result, a generation count of 50 was identified for subsequent optimisation process and model executions.

With the generation count number determined, the investigation then looked into the population size and the smallest population that could maintain a consistent performance for reaching the search outcomes. From the results of population sizes of 60, 70, 80 and 90, we observed that the convergence performance for a population of 60 and 90 showed the best convergence performance. We saw similar convergence performances, as well as consistent final convergent results. While a population size of 70 also shows consistent final convergent points, the performance was not as good as that found with population sizes of 60 and 90. The convergent performance for a population size of 80 was also not as good as that observed for 60 and 90.

Based on these observations and the requirement to minimise the population size as reasonably as possible, a population size of 60 and generation count of 50 was identified for use for subsequent case study experimentation.

After confirming the behaviour and performance for each of the Objective Function and solution search algorithms individually, the last test conducted was to determine how well the complete optimisation processing chain predicts and optimises, and how those outcomes compare with the measured results. Using the same test arrangements and results found for the Objective Function testing, an analysis was conducted to firstly see how well the average measured critical chain latencies aligned with the predicted latencies across the generations. Secondly, the analysis looked at how the results of the measured latency converged across the generations created, and how that convergence performance aligned with the predicted results convergence.

Figure 6.19 shows one pair of convergence comparison results between the measured and predicted latencies from a series of 6 test results and analyses across 3 different network configurations. Appendix D.6 shows the results for the other 5 pairs of test and analysis results.

From these results, it can be seen that both the measured results and predicted results converge and the convergence is in phase. Once again, it can be observed that the predicted times are constantly greater than the measured times. While we can see that the measured results do not converge to the smallest measured results, the actual difference is relatively small, with most differences being less than 10 milliseconds and the largest in the order of 25 milliseconds.

The cause behind this could be a combination of the network jitters, time drift and precision of time synchronisation across the distributed computing nodes. Future research would look to further investigate and quantify the effect of jitter, as well as aim to construct methods to minimise the differences through improved time modelling, alongside improved distributed time synchronisation methods.

Building on all these tests, results and analysis, it can be concluded with reasonable confidence that the combined behaviour and performance of the Objective Function algorithm and search algorithm delivers a reasonable level of accuracy. As a result, optimal solutions are identified that ensure software deployment profiles that lead to delivery of the desired system performances.

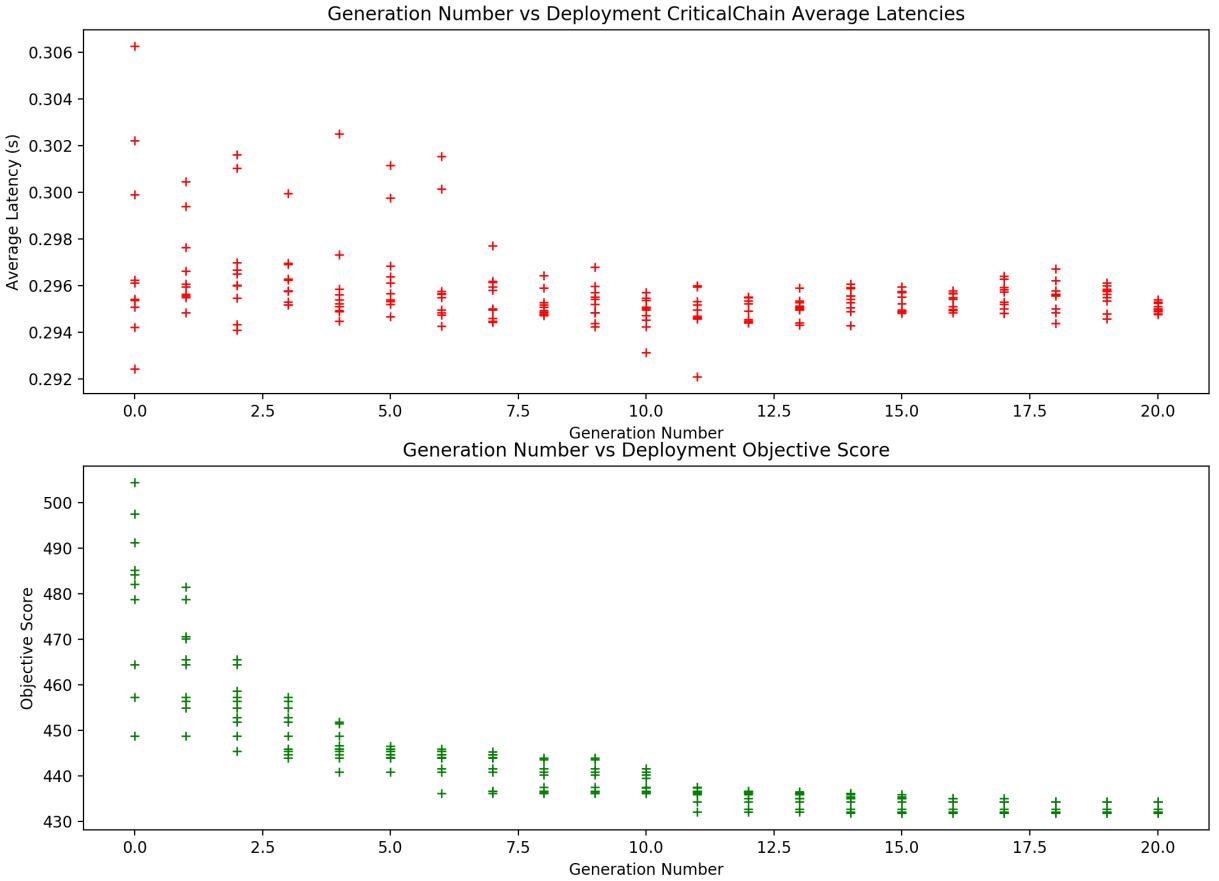

Figure 6.19: Latencies predicted versus measured convergence: 100Mb Network Configuration

6.4 Interface and String Latency Optimisation

As detailed in the previous chapter (Chapter 5), the Objective Function and search process looks for solutions that satisfy predefined temporal performance requirements for either a segment of the critical thread (string) or a single connection. Using modelled latency requirements, the temporal performance of the critical chain is minimised only to a point where string and single connection temporal requirements are also satisfied.

To establish how well the Objective Function performs at satisfying any temporal performance requirement, a second test model was deployed. Building on the first test model, this model included constraints for temporal performance requirements for a string maximum latency of 130ms and an interface (single connection) maximum latency of 60ms. No constraints for resource utilisation or deployment were present.

To undertake this testing and subsequent analysis, the Testing Framework also included the ability to extract out raw (i.e., not including the final objective function satisfaction adjustment) string and interface latency predictions.

Figure 6.20 and Figure 6.21 shows the predicted string latencies and measured average string latencies for the deployed test model. From these plots we observe that the algorithm is converging with solutions identified across the generations sitting around the required 130ms, with a range from approximately 126ms to 148ms.

While a comparison of the results from the predicted and average measured

string latencies continues to indicate a good level of prediction performance, there is a need for the prediction algorithm to behave differently when satisfying temporal requirements. This change leads to all predictions and optimised solutions delivering latency performances that do not exceed any required maximum latency. To achieve this, the tuning applied to the time representation and prediction algorithm for temporal performance ensures less optimistic predictions, so the measured results are either the same or less.

As a result, the modeller and analysis can work on the principle that the predictions are going to meet or perform better than the requirements defined in the model.

Figure 6.22 shows the results of the largest latencies for each population member across the generations. Comparing these results with the prediction results (Figure 6.20) shows that the predictions are also too optimistic and are consistently better than the measured largest latencies.

Figure 6.23 and Figure 6.24 show the results of a predicted temporal performance and the measured interface latency for each population member across the generations. Once again, the predictions are too optimistic with the measured results constantly greater [5].

In response to this, individual adjustment factors were applied to the prediction algorithm for the string and interface temporal performance prediction to ensure less optimistic algorithm performance. To minimise complexity with the first

[5] It should be noted that while the defined temporal performance requirements indicate a need to be equal or less than 130ms for the string and 60ms for the interface, neither the prediction nor the measured results were able to achieve this

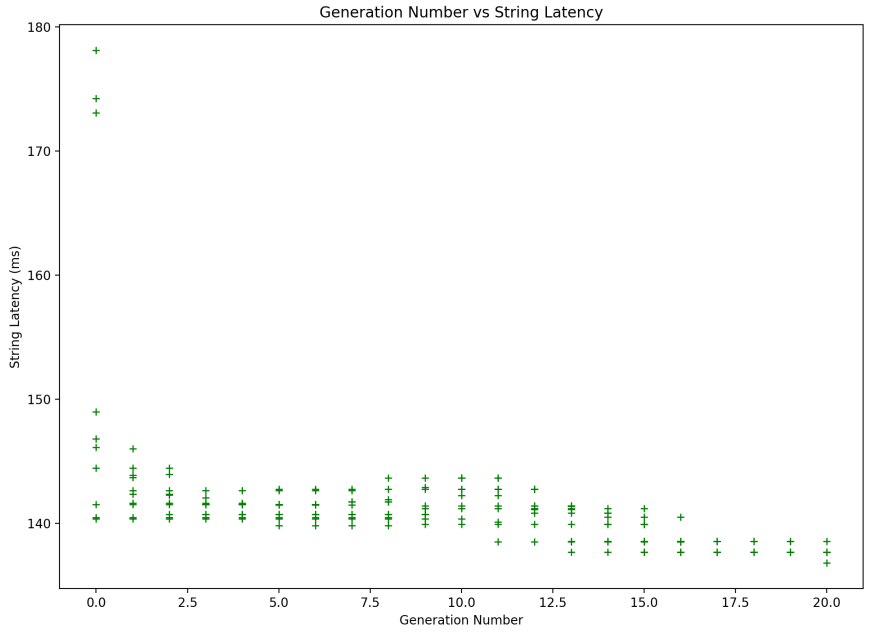

Figure 6.20: Predicted string latencies for a 130ms modelled defined constraint

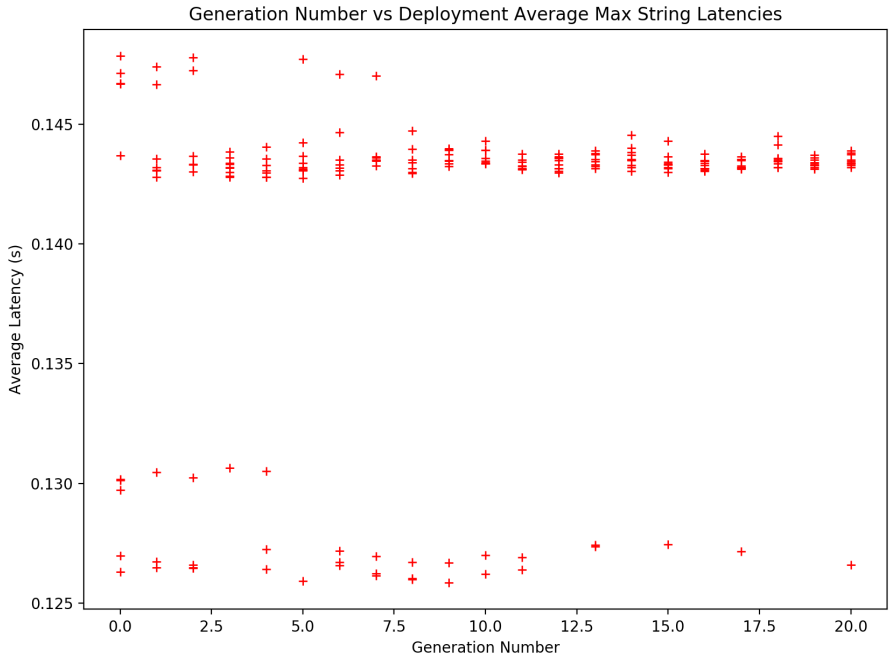

Figure 6.21: Average measured string latencies for a 130ms modelled defined constraint

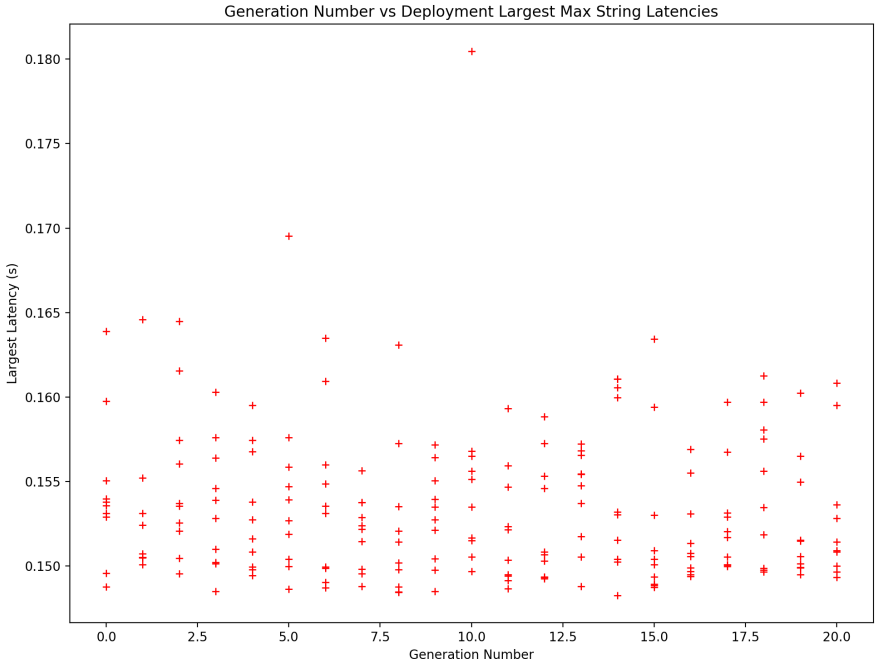

Figure 6.22: Largest measured string latencies for a 130ms modelled defined constraint

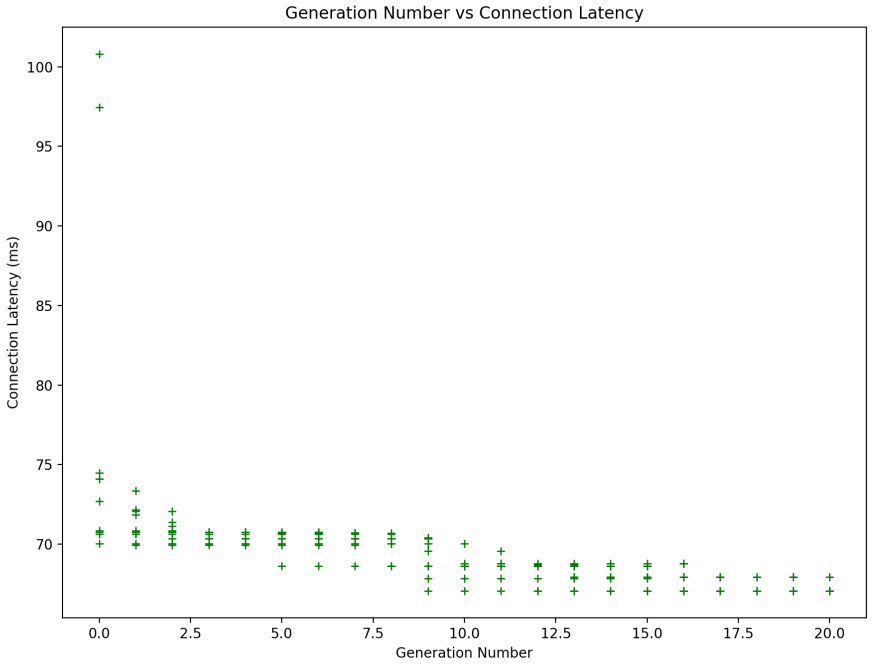

Figure 6.23: Predicted interface latencies for a 60ms modelled defined constraint

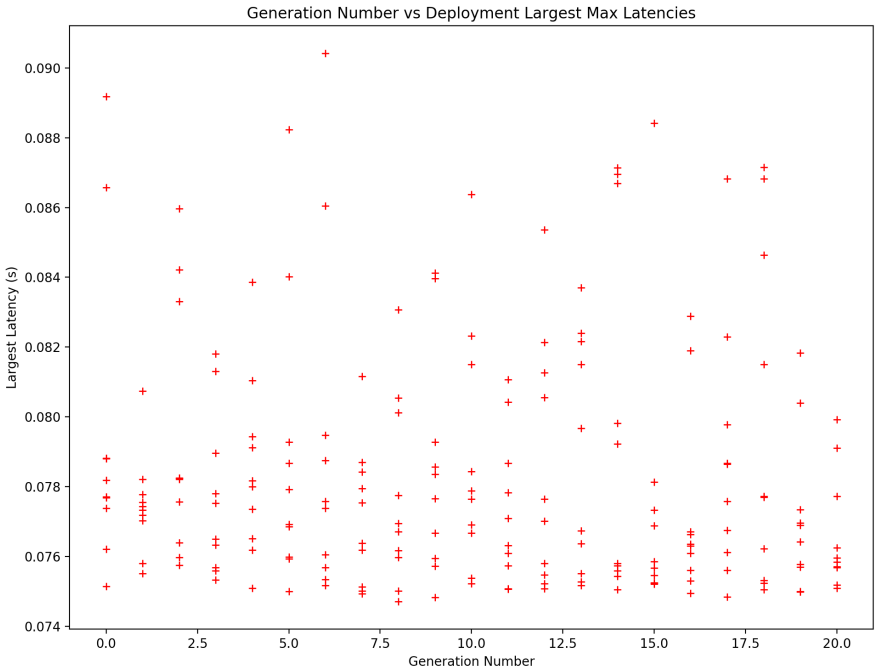

Figure 6.24: Largest measured interface latencies for a 60ms modelled defined constraint

instance of creating such factors, a straightforward approach of accounting for the worst case performance was used. This took the form of simply identifying the largest difference for each population across generations, and then taking the largest from those sets. Future research efforts will look to improved methods for creating the factors around trends and major influences on performance, and possibly be a set of factors across the spectrum of results, rather than a single blanket factor.

From Table 6.3 and Table 6.4, the adjustment factors for string latency predictions and interface latency predictions were set to -26.42% and -30.16%. Using these adjustments a second run of the string and connection latency constraint test model was conducted (Figures 6.25, 6.26, 6.27 and 6.28).

From the prediction plots, we observe that the adjustment factor improves the convergence performance of the prediction algorithm. The accompanying measured latency results for both string and interface also showed improvement. From each graph we see improved convergence performance, as well as the formation of a threshold. While not as clean as desired, we see the solutions are in alignment with the desired behaviour of identifying solutions at the required performance level or better.

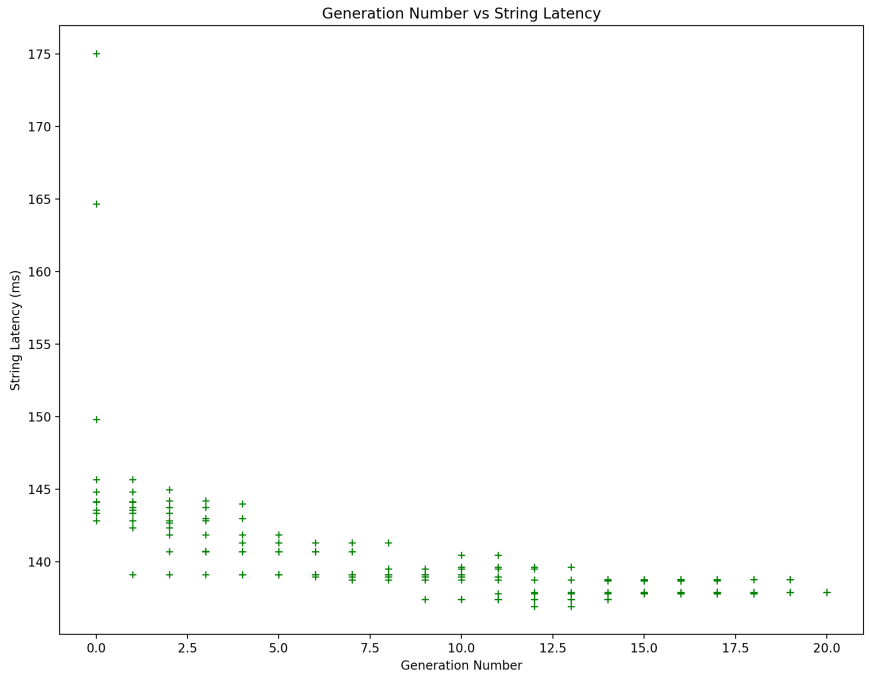

Figure 6.25: Adjusted predicted string latencies for a 130ms modelled defined constraint

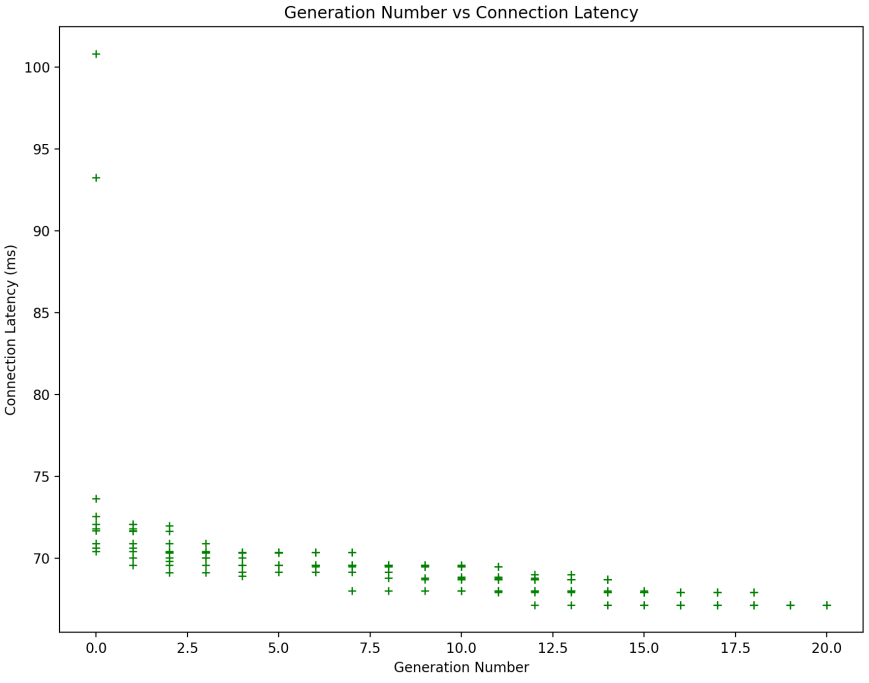

Figure 6.26: Adjusted predicted interface latencies for a 60ms modelled defined constraint

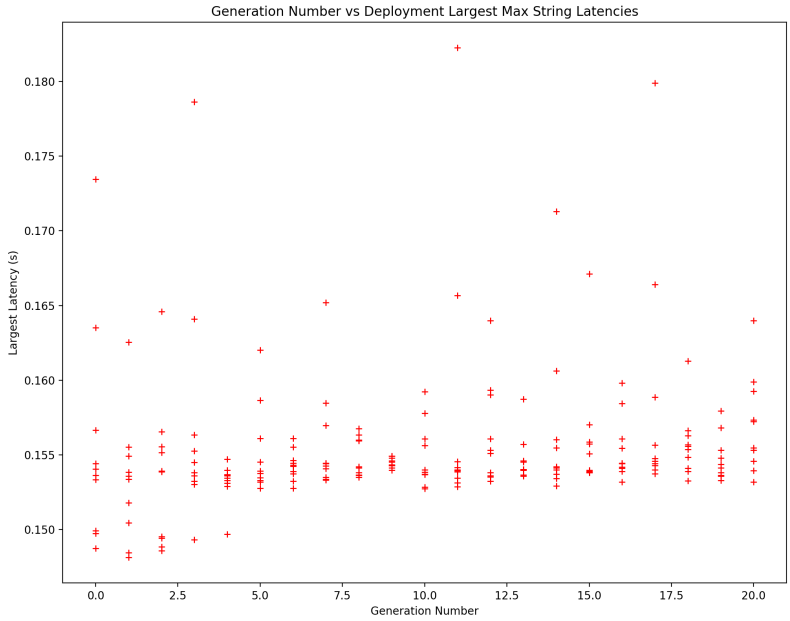

Figure 6.27: Largest measured string latencies with adjustment factor applied for a 130ms modelled defined constraint

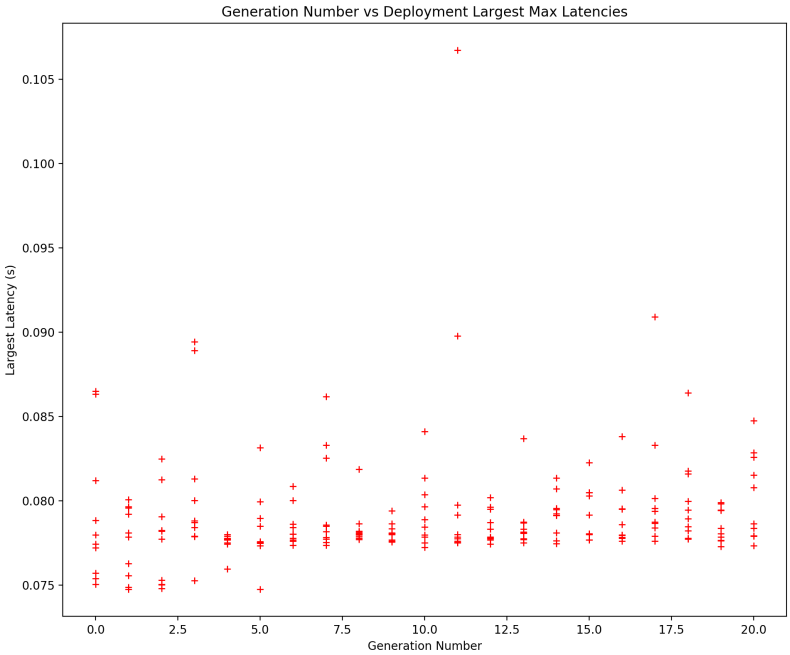

Figure 6.28: Largest measured interface latencies with adjustment factor applied for a 60ms modelled defined constraint

Generation	Population Number	Maximum Latency Difference (%)
0	9	-16.70
1	11	-15.39
2	8	-13.87
3	4	-11.38
4	11	-11.83
5	11	-18.84
6	27	-16.20
7	52	-11.32
8	81	-19.45
9	35	-10.21
10	102	-26.42
11	101	-13.86
12	130	-12.55
13	133	-14.20
14	146	-16.26
15	146	-17.97
16	139	-13.25
17	173	-15.27
18	160	-17.13
19	160	-16.39
20	185	-16.81
	Largest Delta	-26.42

Table 6.3: Maximum Differences between Predicted and Measured Maximum String Latencies

6.5 Software Component (and workload) Deployment Optimisation

Building on the desired software deployment performance we observed during the critical thread prediction performance validation (Figure 6.11 and Figure 6.12), a test was conducted to confirm the optimisation algorithm's ability to satisfy the

Generation	Population Number	Largest Difference Interface (%)
0	9	-26.28
1	11	-15.42
2	8	-18.86
3	4	-15.637
4	17	-18.53
5	11	-26.14
6	27	-28.52
7	27	-15.37
8	79	-18.08
9	57	-22.57
10	102	-17.75
11	106	-17.86
12	130	-24.23
13	133	-23.64
14	146	-28.28
15	146	-30.16
16	139	-22.02
17	173	-27.82
18	173	-27.82
19	178	-20.47
20	175	-19.163
	Largest Difference	-30.16

Table 6.4: Largest Differences between Predicted and Measured Maximum Interface Latencies

modelled CPU utilisation requirements. Once again, utilising the initial test model, CPU utilisation requirements were modelled for each available test computing node. These requirements were as follows [6]:

- Cranberry01 -> Nil spare CPU capacity,
- Cranberry02 -> 60% spare CPU capacity,

[6]It should be also noted the spare CPU utilisation capacities are from the hardware threshold level of 80% detailed in Section 5.2.6

- Mandarin01 -> 60% spare CPU capacity,
- Mandarin02 -> 40% spare CPU capacity.

Utilising the CPU utilisation measurements found within the MEDEA logging framework, analysis was conducted to determine the worst case CPU utilisation within each deployment solution within a generation. Following this, the maximum utilisation for each generation was extracted to produce an envelope curve for the maximum CPU utilisation for each computing node. These envelopes, and more importantly the final utilisation, were then compared with the modelled spare CPU utilisation requirements.

Figure 6.29, 6.30, 6.31 and 6.32 show the maximum CPU utilisation for each deployment solution identified within each generation, across generations. While Figure 6.33, 6.34, 6.35 and 6.36 shows the envelope curve for maximum spare utilisation in comparison with the required spare CPU utilisation and hardware (threshold) CPU utilisation for each computing node.

From the envelope curves we observe that the algorithm performs reasonably well in satisfying each spare CPU utilisation requirement established in the test model. We see three of the maximum CPU utilisation exhibiting levels under the desired threshold (in the case of *mandarin01*, well under, at essentially zero), while the envelope for *cranberry02* falls slightly above the required threshold as a result of the granularity of the workloads (equally) deployed within this test model (Subsection 6.1).

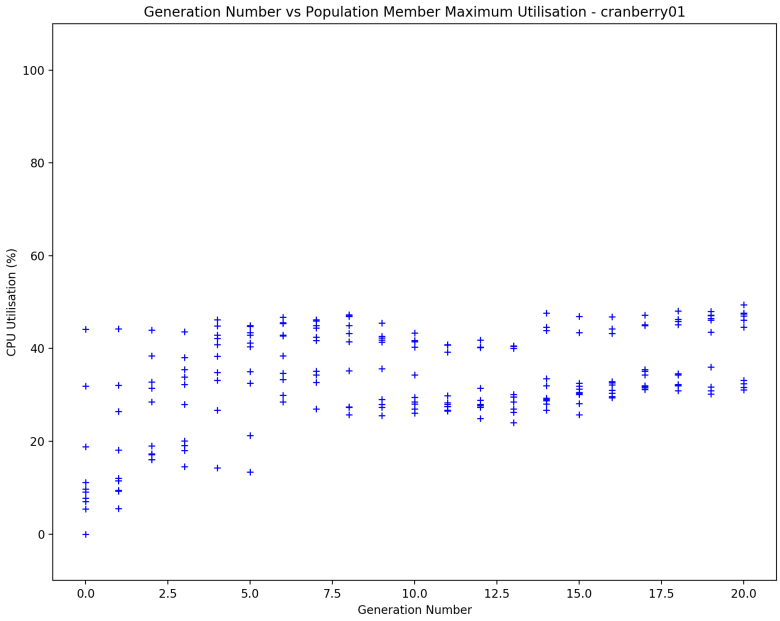

Figure 6.29: CPU Utilisation for each deployment within the generation for computing node *cranberry01*, requiring no spare CPU resources

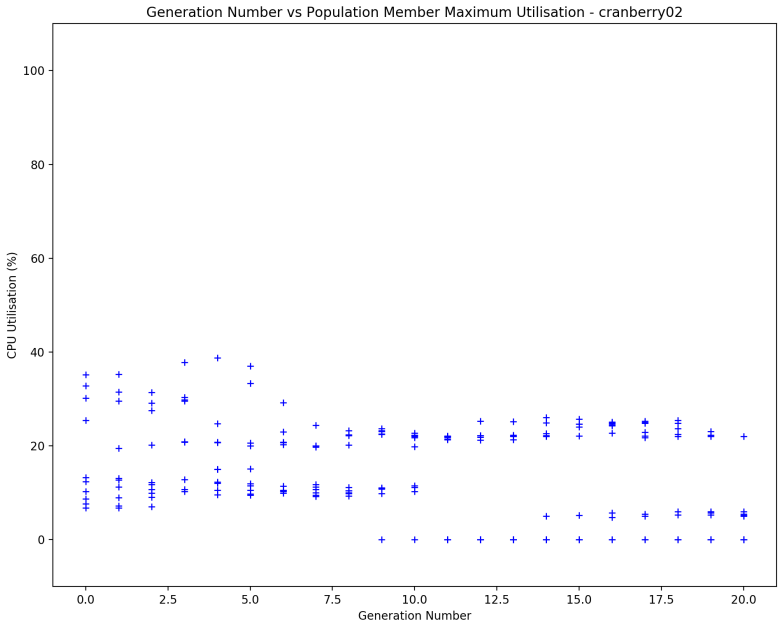

Figure 6.30: CPU Utilisation for each deployment within the generation for computing node *cranberry02*, requiring 80% total spare CPU resources

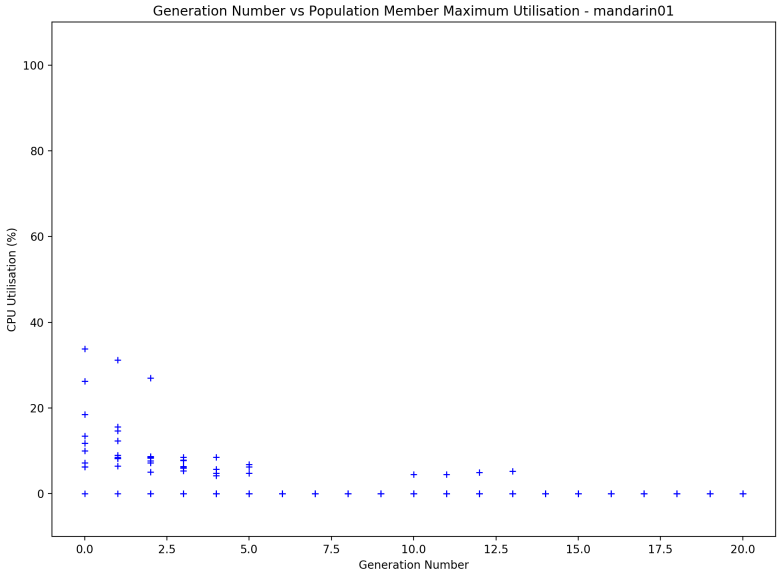

Figure 6.31: CPU Utilisation for each deployment within the generation for computing node *mandarin01*, requiring 80% total spare CPU resources

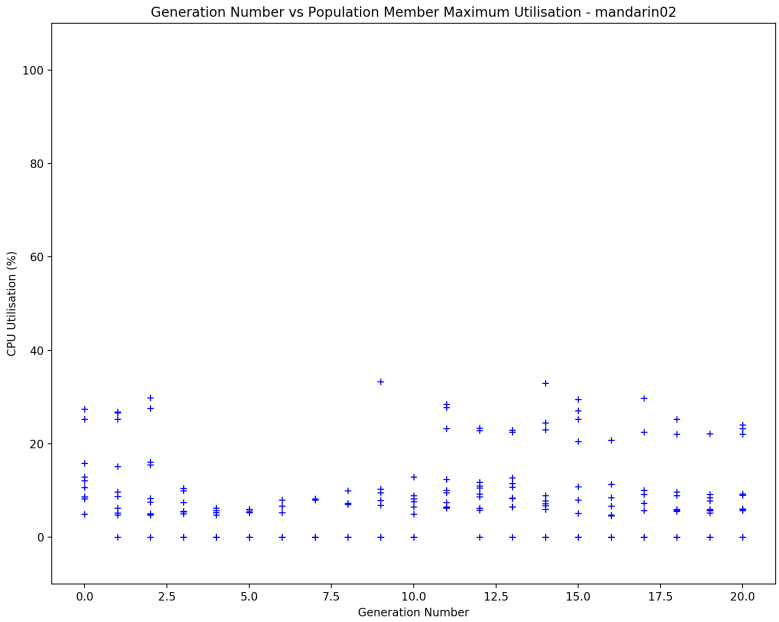

Figure 6.32: CPU Utilisation for each deployment within the generation for computing node *mandarin02*, requiring 60% total spare CPU resources

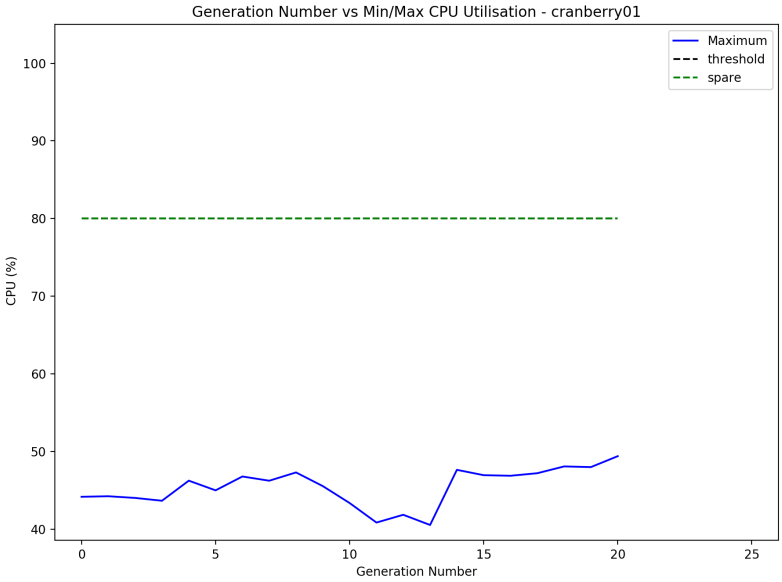

Figure 6.33: Maximum CPU Utilisation for generation for computing node *cranberry01*, requiring no CPU resources

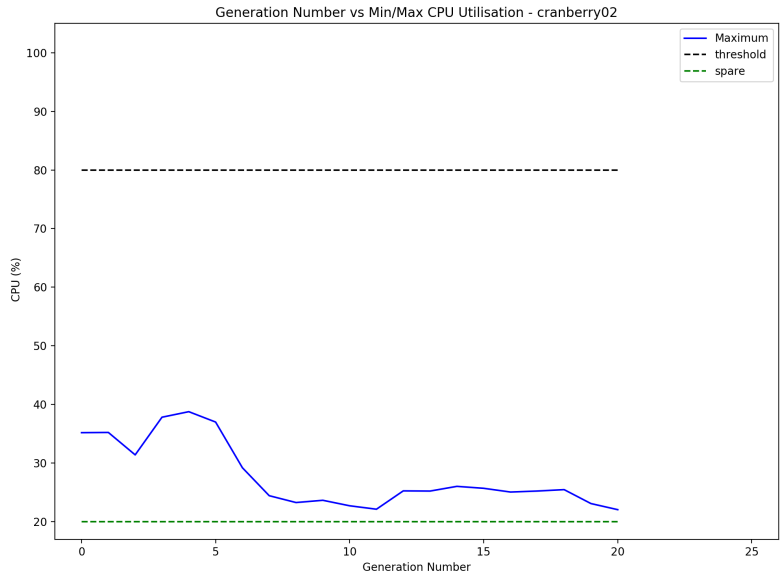

Figure 6.34: Maximum CPU Utilisation for each generation for computing node *cranberry02*, requiring 80% total spare CPU resources

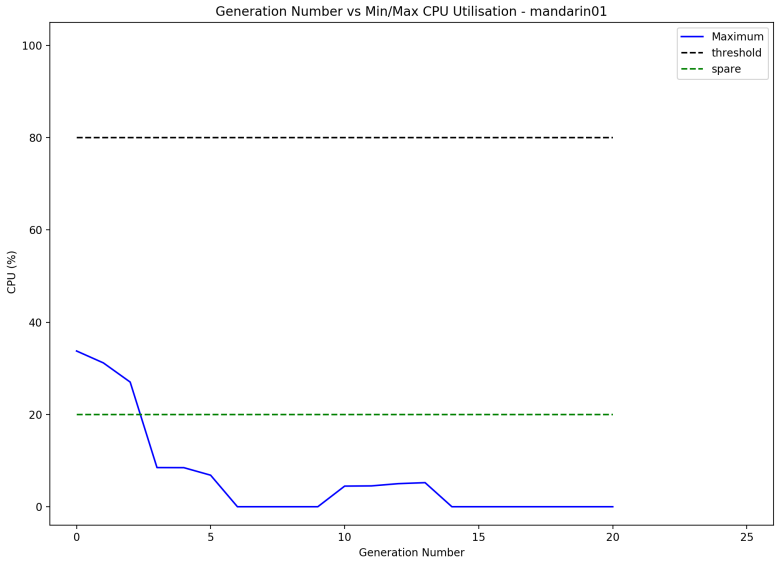

Figure 6.35: CPU Utilisation for each generation for computing node *mandarin01*, requiring 80% total spare CPU resources

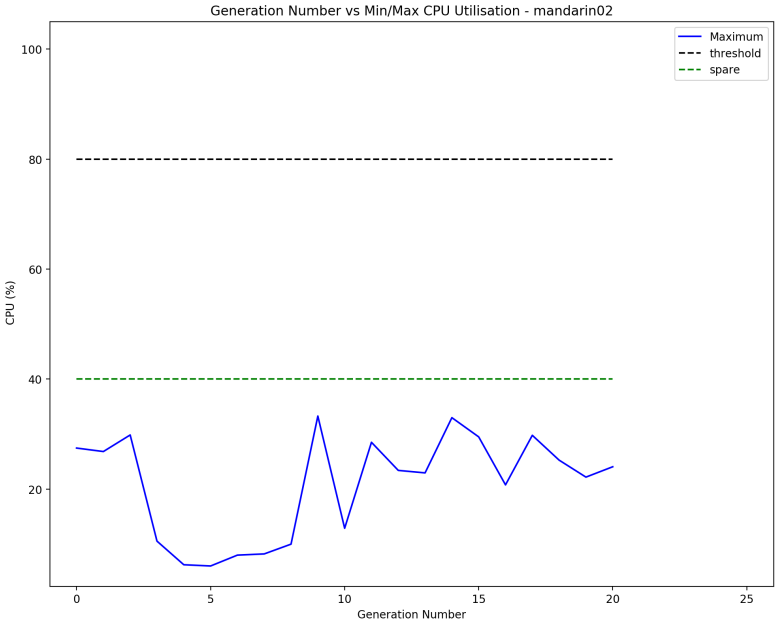

Figure 6.36: CPU Utilisation for each generation for computing node *mandarin02*, requiring 60% total spare CPU resources

6.6 Software Component Mandate Deployment Confirmation

Following the testing and validation efforts conducted on the various components of the optimisation algorithm, the last algorithm validation task conducted established that the solutions are abiding by any software component mandate deployment constraint declared. This was achieved by constructing text files and simply comparing the deployment arrays generated to confirm the final generations do not violate any mandated software deployment requirement.

Table 6.5 shows the details from the deployment arrays of each member of the final generation for the test model used for the latency and string testing effort (Subsection 6.4). The array position represents a particular software component, while the number at the array position represents the node the software component has been deployed to (node numbers ranging from zero to three to represent four nodes available for deployment). The subsequent tables (Table 6.6, Table 6.7 and Table 6.5) show the details of the component deployment array of each member of the final generation for three test models modified to have a *componentMandateDeployGroup* requirement modelled, a *componentMandateDeployOnly*[7] requirement modelled and a *noColocationPairId* requirement modelled accordingly, and separately.

[7]This new constraint has been introduced to account for the experimentation environment constraints and requirements of the case study scenarios to come. This constraint allows for a single software component to be deployed to a particular computing node and only that software component

In the case of the *componentMandateDeployGroup* constraint, this was represented as the *Sender_1* component (first array position on the software component deployment array) being deployed to *Cranberry01* (node number zero). In the case of the *componentMandateDeployOnly* constraint, this was represented as the *Receiver2_3_2* component (the last array position on the software component deployment array) being deployed to *Mandarin02* (node number three). Finally, the *noColocationPairId* case was represented by *Receiver2_2_1* component (software component deployment array position seven) and *Receiver2_2_2* component (software component deployment array position nine), which cannot have the same node numbers. Complete copies of these internal result text files can be found in Appendix D.7.

Generation	Population Number	Software Component Deployment Array
20	123	[3, 0, 1, 1, 1, 2, 2, 1, 2, 1]
20	152	[3, 0, 1, 1, 1, 2, 2, 1, 2, 1]
20	175	[3, 0, 1, 1, 1, 2, 2, 1, 2, 1]
20	179	[3, 0, 1, 1, 1, 2, 2, 1, 2, 1]
20	187	[3, 0, 1, 1, 1, 2, 2, 1, 2, 1]
20	194	[3, 0, 1, 1, 1, 2, 2, 1, 2, 1]
20	195	[3, 0, 1, 1, 1, 2, 2, 1, 2, 1]
20	201	[3, 0, 1, 1, 1, 2, 2, 1, 2, 1]
20	206	[3, 0, 1, 1, 1, 2, 2, 1, 2, 1]
20	208	[3, 0, 1, 1, 1, 2, 2, 1, 2, 1]

Table 6.5: Test Model with no mandate deployment constraint modelled

Generation	Population Number	Software Component Deployment
20	136	[0, 1, 2, 2, 2, 3, 2, 3, 3, 2]
20	198	[0, 1, 2, 2, 2, 3, 2, 3, 3, 2]
20	207	[0, 1, 3, 2, 2, 2, 2, 3, 3, 2]
20	157	[0, 1, 3, 2, 3, 3, 3, 2, 2, 2]
20	192	[0, 1, 3, 2, 3, 3, 3, 2, 2, 2]
20	206	[0, 1, 3, 2, 3, 3, 3, 2, 2, 2]
20	147	[0, 1, 3, 2, 2, 3, 2, 3, 3, 2]
20	178	[0, 1, 3, 2, 2, 3, 2, 3, 3, 2]
20	190	[0, 1, 3, 2, 2, 3, 2, 3, 3, 2]
20	149	[0, 1, 3, 2, 3, 2, 3, 2, 2, 2]

Table 6.6: Test Model with a *componentMandateDeployGroup* mandate deployment constraint modelled

Generation	Population Number	Software Component Deployment Array
20	144	[0, 2, 1, 1, 1, 1, 1, 1, 1, 3]
20	175	[0, 2, 1, 1, 1, 1, 1, 1, 1, 3]
20	189	[0, 2, 1, 1, 1, 1, 1, 1, 1, 3]
20	207	[0, 2, 1, 1, 1, 1, 1, 2, 1, 3]
20	162	[0, 2, 1, 1, 1, 1, 0, 1, 1, 3]
20	184	[0, 2, 1, 1, 1, 1, 0, 1, 1, 3]
20	156	[0, 2, 1, 0, 1, 1, 1, 2, 1, 3]
20	201	[0, 2, 1, 0, 1, 1, 1, 2, 1, 3]
20	84	[0, 2, 1, 1, 1, 2, 1, 1, 1, 3]
20	100	[0, 2, 1, 1, 1, 2, 1, 1, 1, 3]

Table 6.7: Test Model with a *componentMandateDeployOnly* mandate deployment constraint modelled

We observe the *componentMandateDeployGroup* model results show the num-

Generation	Population Number	Software Component Deployment Array
20	200	[3, 1, 2, 2, 2, 0, 2, 0, 0, 2]
20	206	[3, 1, 2, 2, 2, 0, 2, 0, 0, 2]
20	99	[3, 1, 2, 2, 0, 2, 2, 0, 0, 2]
20	122	[3, 1, 2, 2, 0, 2, 2, 0, 0, 2]
20	127	[3, 1, 2, 2, 0, 2, 2, 0, 0, 2]
20	174	[3, 1, 2, 2, 0, 2, 2, 0, 0, 2]
20	203	[3, 1, 2, 2, 0, 2, 2, 0, 0, 2]
20	153	[3, 1, 2, 2, 0, 0, 2, 0, 0, 2]
20	183	[3, 1, 2, 2, 0, 0, 2, 0, 0, 2]
20	133	[3, 1, 0, 2, 0, 2, 2, 0, 0, 2]

Table 6.8: Test Model with a *noColocationPairId* mandate deployment constraint modelled

ber zero ('Cranberry01') is only used for the 'Sender_1'component [8], while the results for *componentMandateDeployOnly* show that the *Receiver2_3_2* component is only deployed to node number three (*Mandarin02*). Lastly, the *noColocationPairId* results show the software components found at array positions seven and nine are never deployed to the same node number.

A final confirmation was to execute a test model with all three mandate deployment constraints modelled, to ensure the solution identified from the optimisation process can simultaneously satisfy more than one modelled mandate deployment constraint. Table 6.9 shows the results from this mandate deployment multi-constraint model.

[8] in this case node number zero is only deployed once in the final generation, but throughout the generation it can be seen number zero is used for deployment across other components, which is a valid response

Generation	Population Number	Software Component Deployment Array
20	135	[0, 1, 2, 2, 2, 2, 2, 2, 0, 3]
20	141	[0, 1, 2, 2, 2, 2, 2, 2, 0, 3]
20	143	[0, 1, 2, 2, 2, 2, 2, 2, 0, 3]
20	157	[0, 1, 2, 2, 2, 2, 2, 2, 0, 3]
20	163	[0, 1, 2, 2, 2, 2, 2, 2, 0, 3]
20	178	[0, 1, 2, 2, 2, 2, 2, 2, 0, 3]
20	182	[0, 1, 2, 2, 2, 2, 2, 2, 0, 3]
20	184	[0, 1, 2, 2, 2, 2, 2, 2, 0, 3]
20	191	[0, 1, 2, 2, 2, 2, 2, 2, 0, 3]
20	197	[0, 1, 2, 2, 2, 2, 2, 2, 0, 3]

Table 6.9: Test Model with mandate deployment multi-constraints modelled

We see from these results that across this final generation the optimised solutions do satisfy three constraints modelled in this test. As a result, this indicates the optimisation process is capable of reconciling more than one deployment constraint in any one execution and therefore can produce multi-constraint optimisation solutions.

7. Deployment Optimisation Case Studies

As detailed in Chapter 1, an undersea sensor network system is one example of a constrained SoS. It consists of numerous computing platforms, geographically distributed over a designated investigation area, and is required to collect data, store and transmit to a central collection point (Jindal, Saxena, and Singh, 2014 and Ovaliadis, Savage, and Kanakaris, 2010). Undersea sensor network systems have numerous applications within areas such as environmental monitoring, defence, navigation and leisure (Felemban et al., 2015).

As highlighted by Akyildiz, Pompili, and Melodia, 2004, Awan et al., 2019, Heidemann et al., 2006 and Ovaliadis, Savage, and Kanakaris, 2010, an undersea sensor network system design has to consider constrained and fixed resources. Furthermore, its deployment has to account for non-functional requirements such as limited bandwidth, temporal performance impacts, significant error rates, changing power demands, power supply constraints, resource utilisation and system failures.

In this chapter, an example of an undersea sensor monitoring network system

design investigation will be used to demonstrate the new software deployment optimisation capability introduced with this thesis. It will demonstrate its ability to search the large and complex solution space for deploying software across available computing nodes, while satisfying different design non-functional requirements. This demonstration will start at a baseline design with no performance optimisation considered, then introduces non-functional requirements across three different scenarios, whereby new non-functional requirements are added with each scenario. Table 7.1 details the non-functional requirements introduced across four scenarios.

Scenario Number	Non-functional Requirements
0	Nil
1	System Failure Resilience
2	System Failure Resilience Power Budget Reductions
3	System Failure Resilience Power Budget Reductions Temporal Performance

Table 7.1: System Design Investigation Scenarios

Furthermore, the three design investigation scenarios considered four design changes:

- **Non-functional Design Change 1: CPU Resource Utilisation Change** for identified nodes within the sensor network. There is a requirement to reduce the node CPU resource utilisation levels (and increase their spare CPU resource capacity) from normal conditions. The aim will be to identify

software deployment profiles that best accommodate the impact of these new CPU utilisation levels.

- **Non-functional Design Change 2: Ensure Redundancy** for identified software component instances. There is a need to ensure redundancy exist within the system network by ensuring identified software components do not reside on the same node. The aim will then be to identify software deployment profiles that accommodate these identified requirements.
- **Non-functional Design Change 3: Latency Performance** for the identified CriticalChain has defined latency performances. These latency requirements will mandate the maximum threshold for the end-to-end timings of transmissions along that critical thread, including for a subsection (or segment) of the critical chain, or a single interface. The aim will then be to identify software deployment profiles that accommodate the impact of ensuring violation of those maximum latency thresholds does not occur.
- **Non-functional Design Change 4: Mandated Software Placement** for identified software component instances. There is a need to ensure components are placed on (deployed to) certain nodes within the undersea sensor system network as a result of system architectural requirements. The aim will be to identify software deployment profiles that accommodate the need to have certain software deployed to certain nodes.

Figure 7.1 depicts the example undersea sensor network system to be investigated. It consists of two seabed sensor nodes, two seawater sensor nodes, a surface sea-buoy node and a land-based receiver station. All the undersea sensors

system nodes and the surface sea-buoy nodes are powered by batteries, while the land based receiver station is connected to the power grid and incurs no power constraints from battery use. Furthermore, to account for environmental influences, the undersea system network communication capabilities will differ. Table 7.2 provides the network interface performance specifications for each node.

Node Number	Node Type	Transmit Rate	Receive Rate
1	Surface Buoy	1Gb	1Gb
2	Seawater Sensor	100Mb	100Mb
3	Seawater Sensor	100Mb	100Mb
4	Seabed Sensor	10Mb	10Mb
5	Seabed Sensor	10Mb	10Mb
6	Land Receiver Station	1Gb	1Gb

Table 7.2: Sensor System Network Transmit and Receive Performance Specifications

We can also see from the connections depicted in Figure 7.1 that not all nodes are directly connected to each other. It is assumed any message received by a node not meant for processing on itself will be forwarded to the intended node through routing configuration. It will also be assumed for experimentation purposes that this forwarding of messages will incur negligible latency.

Figure 7.2 shows the physical experimentation environment used for this case study. Based on a completely virtualised computing environment using VMware[1], the environment consists of 7 virtual machines and a virtual switch. One virtual

[1] https://www.vmware.com

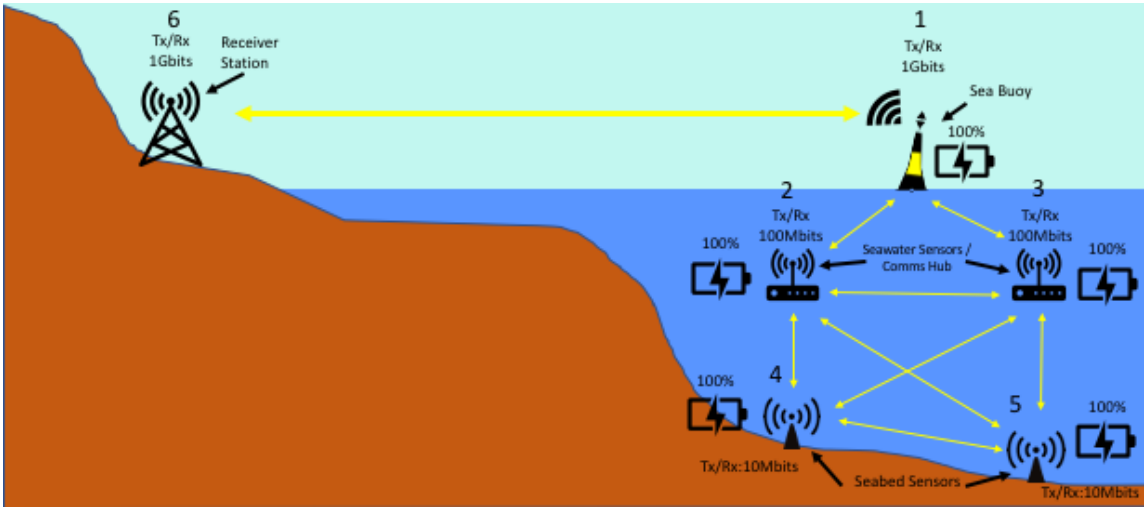

Figure 7.1: Diagram of undersea sensor system network

machine is allocated for the MEDEA Master node, which contains the Jenkins deployment and execution management environment, as well as the MEDEA SQL database. The remaining six virtual machine are allocated as MEDEA run-time nodes, available for software to be deployed onto for execution purposes. For the six run-time nodes, three nodes were CentOS[2] platforms and three were Ubuntu[3] nodes. All six VMware virtual machines were configured with the same resource allocations and all six machines connected to the virtual switch in a simple broadcast switch mode[4].

Lastly, to allow for the representation of the different network/communication bandwidths that would be present within the undersea network system environment, bandwidth throttling capabilities were also introduced. The bandwidth throttling

[2]https://www.centos.org
[3]https://ubuntu.com
[4]Limited configuration was available from the broader experimentation environment

was enabled through the use of WonderShaper[5] software, which was installed on all six run-times nodes.

As a result of the physical arrangement of the experimentation environment, and the fact that there are no occurrences of a single connection between two run-time nodes, a specialised modelling software deployment constraint and solution search was introduced. This would then account for the communication configuration scenario between the "Land Receiver Station" node and the "Surface Buoy" nodes.

Overlaying the physical computing nodes and network configuration of the undersea sensor system is the software system to be deployed in a particular fashion to meet high level non-functional requirements. This software system consists of a set of five software assemblies housing their software component instances and interfaces between them, as well as between assemblies. The five software assemblies are:

- PrimarySeabedSensorAssembly (Figure 7.3)
- SecondarySeabedSensorAssembly (Figure 7.4)
- WaterAssembly_1 (Figure 7.5)
- WaterAssembly_2 (Figure 7.6)
- CommsAssembly (Figure 7.7)

[5]https://github.com/magnific0/wondershaper

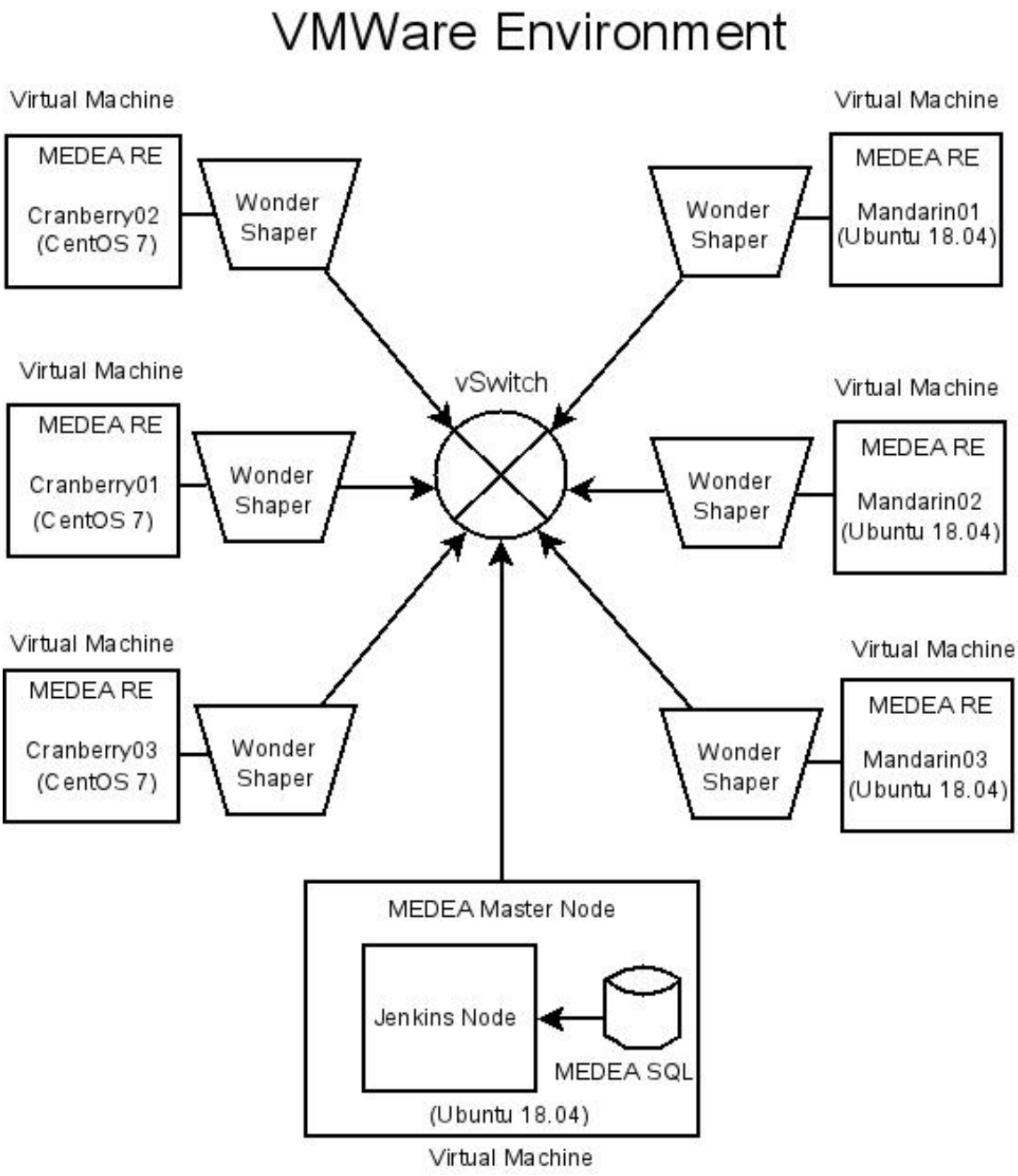

Figure 7.2: VMware-based Experimentation Environment

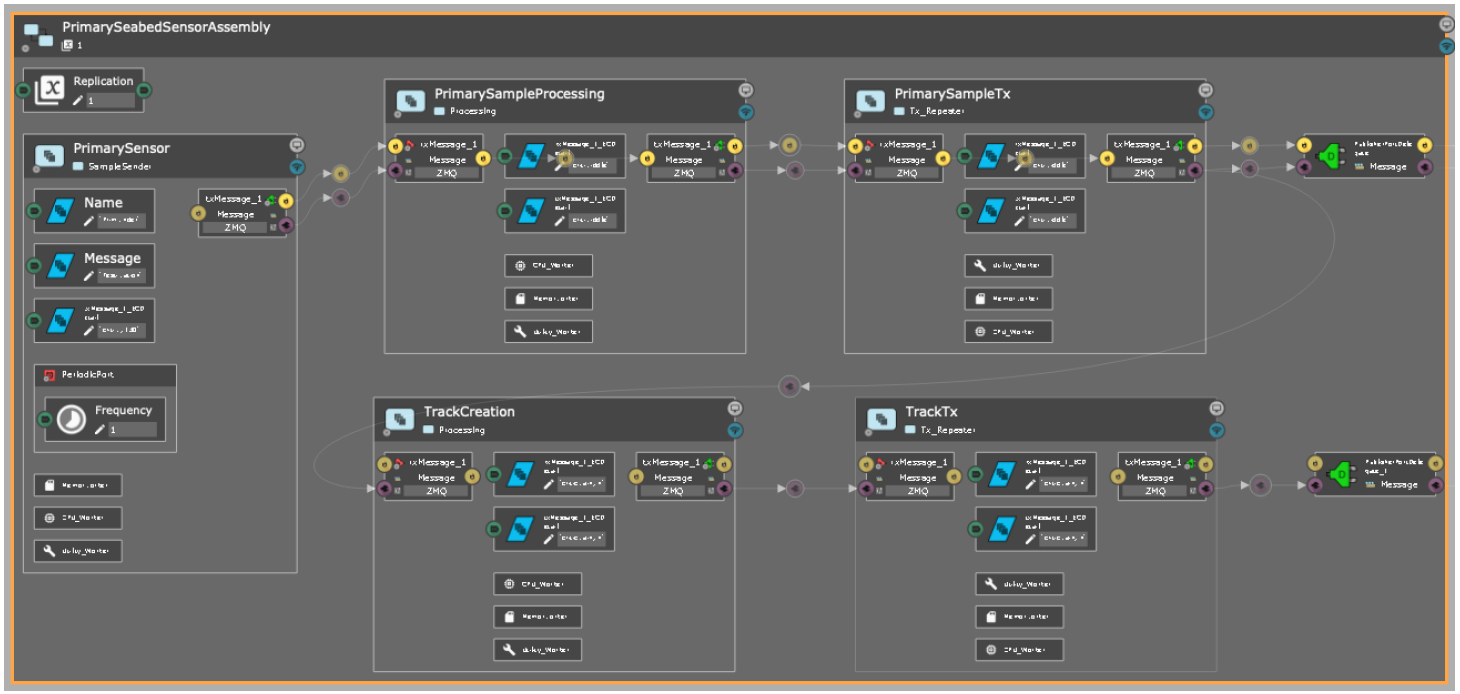

Figure 7.3: MEDEA assembly view of undersea sensor software system PrimarySeabedSensorAssembly

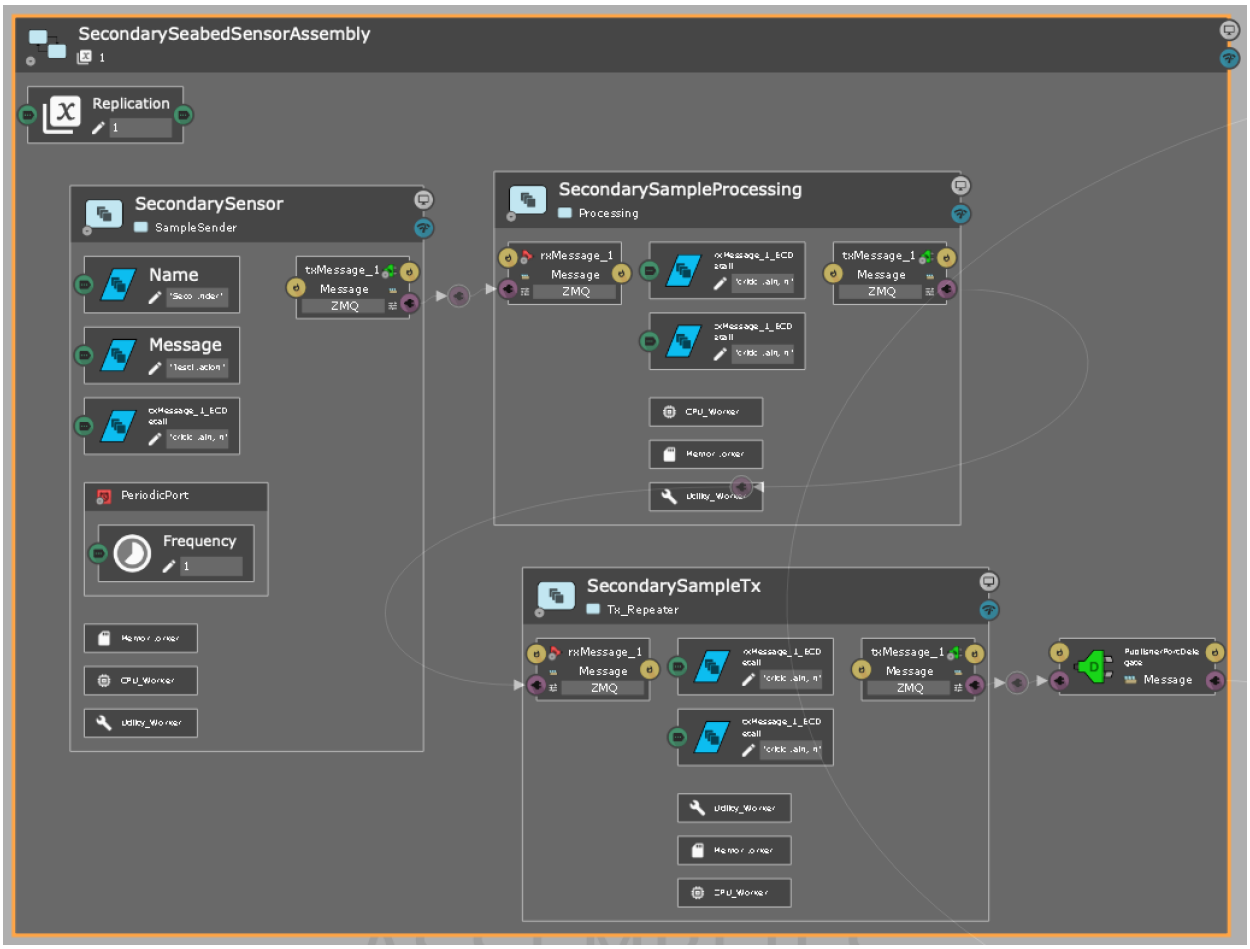

Figure 7.4: MEDEA assembly view of undersea sensor software system SecondarySeabedSensorAssembly

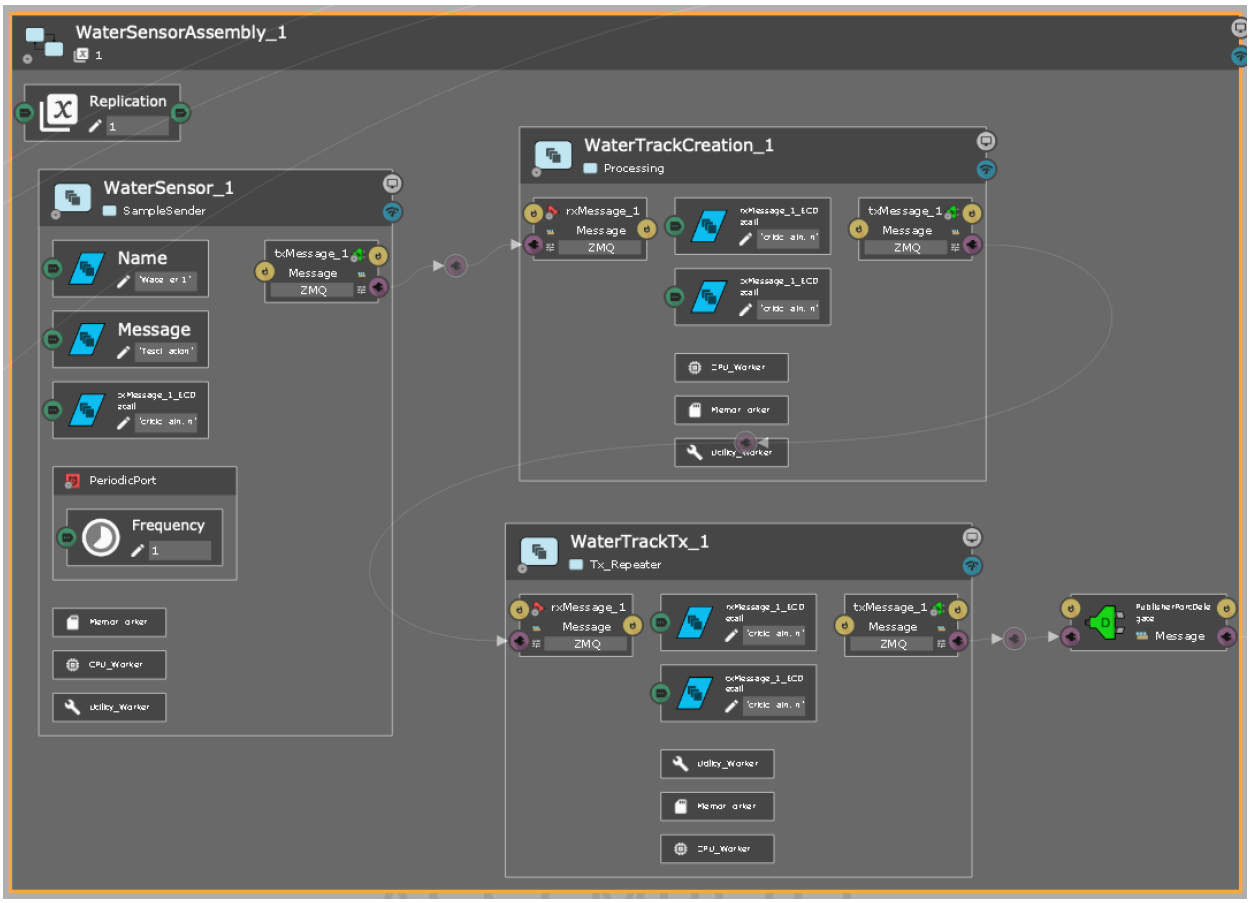

Figure 7.5: MEDEA assembly view of undersea sensor software system WaterAssembly_1 Assembly

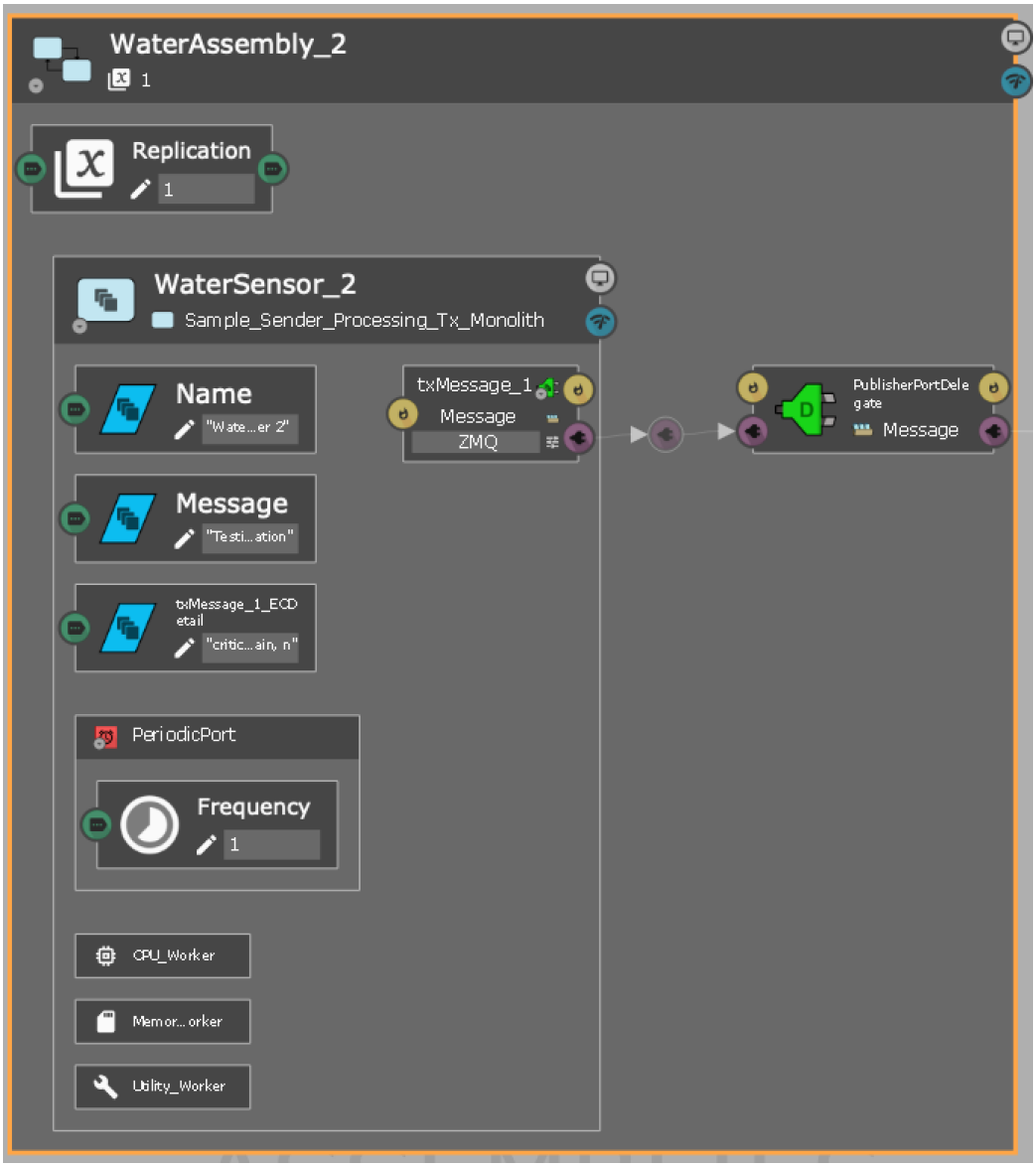

Figure 7.6: MEDEA assembly view of undersea sensor software system WaterAssembly_2 Assembly

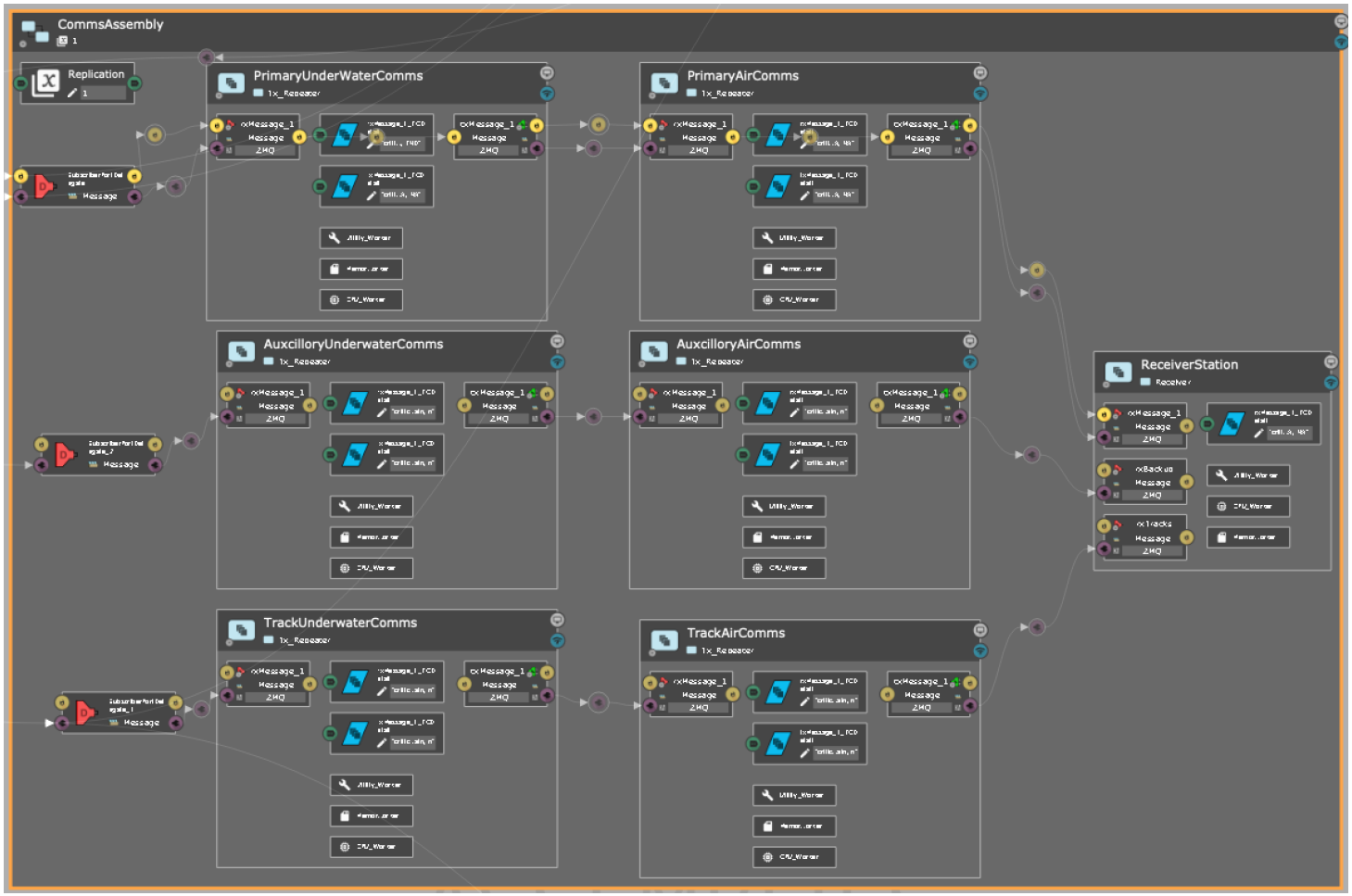

Figure 7.7: MEDEA assembly view of undersea sensor software system CommsAssembly Assembly

Within these assemblies, 19 different software component instances require deployment. These 19 software component instances are based on 5 software component definitions. These definitions are detailed below and diagrams of each can be found in Appendix E.1:

- **SampleSender:** this software component definition represents the sampling of the environment using its sensing capability, representing the sample in message form and the transmission of those messages.
- **Processing:** this software component definition represents the reception of a sample message and the execution of processing on those samples to serve a data processing requirement.
- **Tx_Repeater:** this software component definition represents the reception of messages and transmission onto the sensor system network. This transmission can be via a newly processed and developed message, or be the reception of an existing message from the network and the forwarding of that message.
- **Sample_Sender_Processing_Tx_monolith:** this software component definition represents an internal processing of samples from the environmental sensor, followed by a transmission within the single processing module. It represents a legacy monolithic software system.
- **Receiver:** this software component definition represents the reception of various message types and processing associated with reception of each for correctness and integrity before being stored for use.

Furthermore, in contrast with the MEDEA model developed for testing purposes

in Chapter 6, the component definitions used in this case study consist of workloads of variable sizes.

The entire undersea sensor software system was created using software component instances derived from the software component definitions and connection via the available software component interface, using the assembly interface delegate port. Figure 7.8 shows the MEDEA model assemblies' view of the complete undersea sensor software system architecture.

In addition to the arrangement of software assemblies and component instances, a definition of the critical chain of interest can also be seen. This critical chain can be seen tracing through the *PrimarySeabedSensorAssembly* (Figure 7.3) and *CommsAssembly* (Figure 7.7) assemblies. It starts at the *PrimarySensor* component instance, traces through the *PrimarySampleProcessing*, *PrimarySampleTx* and *PrimaryUnderWaterComms* component instances , and then finishes at the *PrimaryAirComms* component instance.

A closer view of the critical chain can be seen in Figure 7.9.

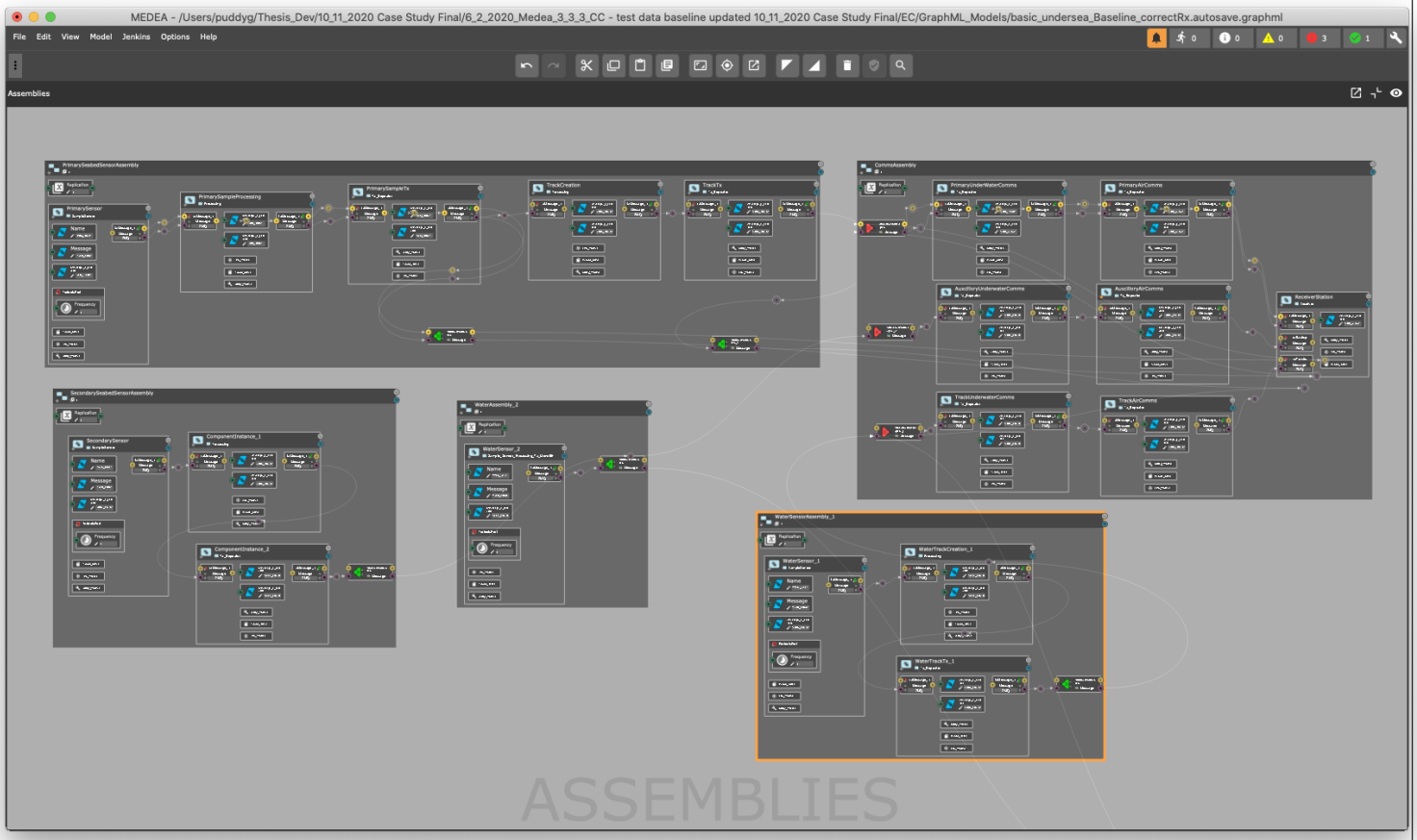

Figure 7.8: MEDEA assembly view of the complete undersea sensor software system

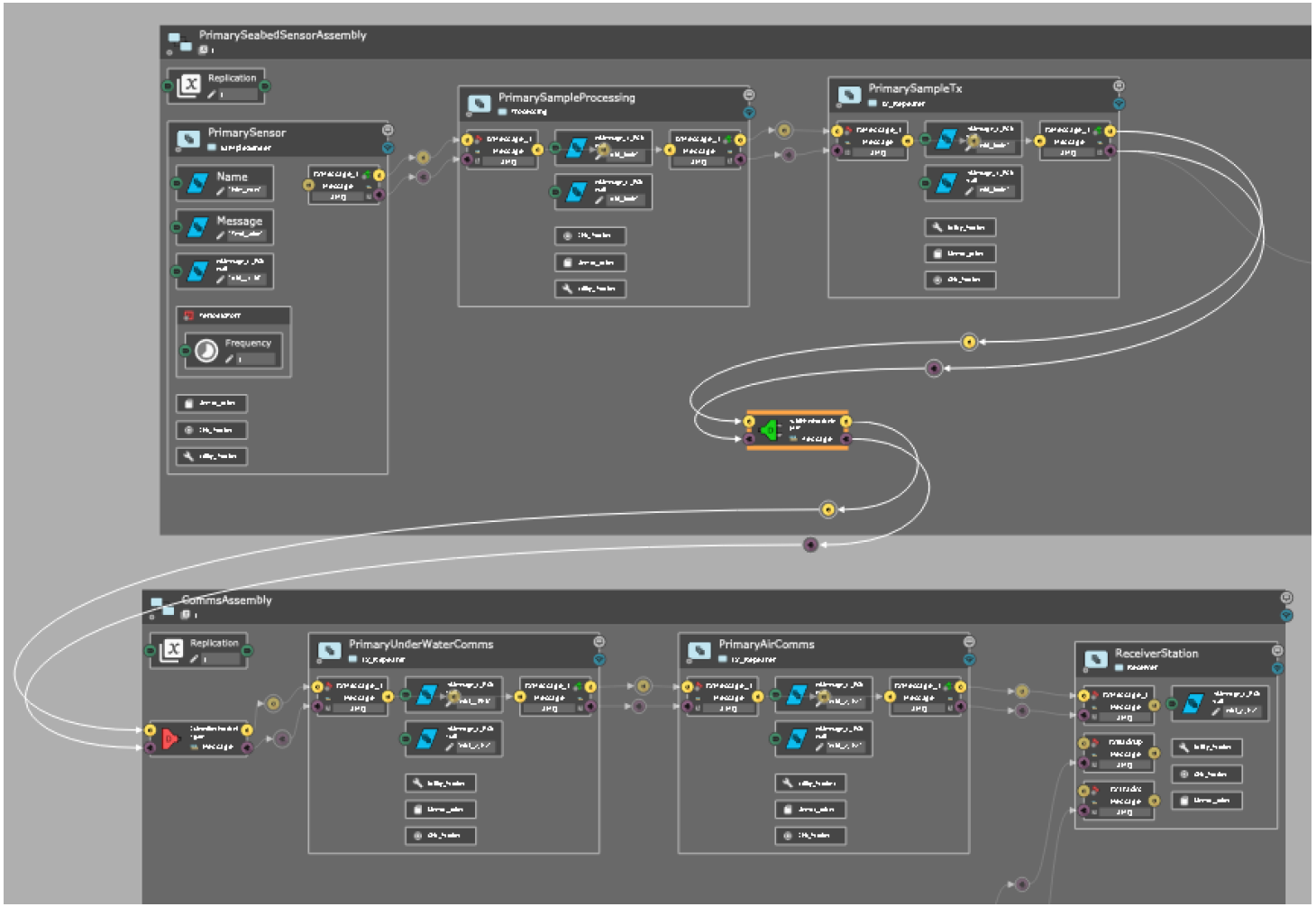

Figure 7.9: MEDEA assembly view of the critical chain for the undersea sensor software system

7.1 Scenario #0: No Constraint Deployment

This initial design scenario provided a baseline of performances where the software deployment is optimised, with no constraints or requirements defined.

The specifications for this baseline design scenario experiment are detailed in Table 7.3.

Configurable Attribute	Specification
Spare CPU Utilisation	Seawater Sensor 2 - 0% Seawater Sensor 3 - 0% Seabed Sensor 4 - 0% Seabed Sensor 5 - 0% Surface Buoy 1 - 0% Land Receiver Station 6 - 0%
Tx/Rx Bandwidth	Seawater Sensor 2 - 100Mb Seawater Sensor 3 - 100Mb Seabed Sensor 4 - 10Mb Seabed Sensor 5 - 10Mb Surface Buoy 1 - 1Gb Land Receiver Station 6 - 1Gb
Population Size	60
Generation Count	50

Table 7.3: Design scenario 0 experimentation conditions

7.1.1 Experimentation Results

In line with the baseline testing conducted in Chapter 6 the first experiment conducted on the case study model was with no performance requirements or constraints modelled. The aim of this experiment was to create a baseline for the

critical chain temporal performance, objective scores and the resource consumption for each available node before the application of any performance requirement or constraint.

The baseline experiment results and the results for the following design scenario investigation are based on the execution of 50 generations, where each generation consists of 60 population members. Within each population member, 58 samples have been taken within a 60 second execution period. Furthermore, each individual population member performance characteristic was either based on an average of the samples or the maximum result from the samples.

Figure 7.10 shows the objective scores converging towards a minimum across the 50 generations. While Figure 7.11 shows the average temporal performance results across the 50 generations for the critical chain identified for the case study model. As observed from the execution and measurement of temporal performance from the previous tests, we see the final generation of measured temporal performances are not the lowest (a delta of less than 10% nonetheless), but the measured performances still exhibits the desired convergence and reduction behaviours across the generations.

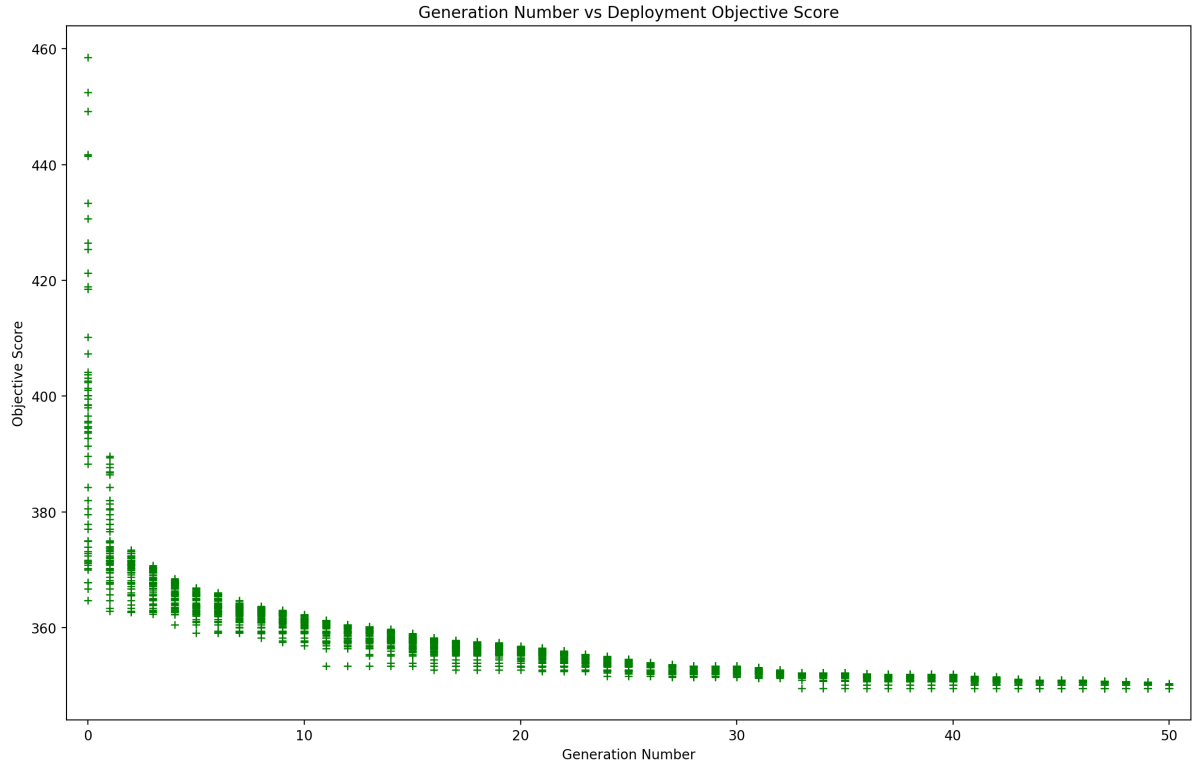

Figure 7.10: Objective scores for an undersea sensor software system across 50 generations and populations of 60

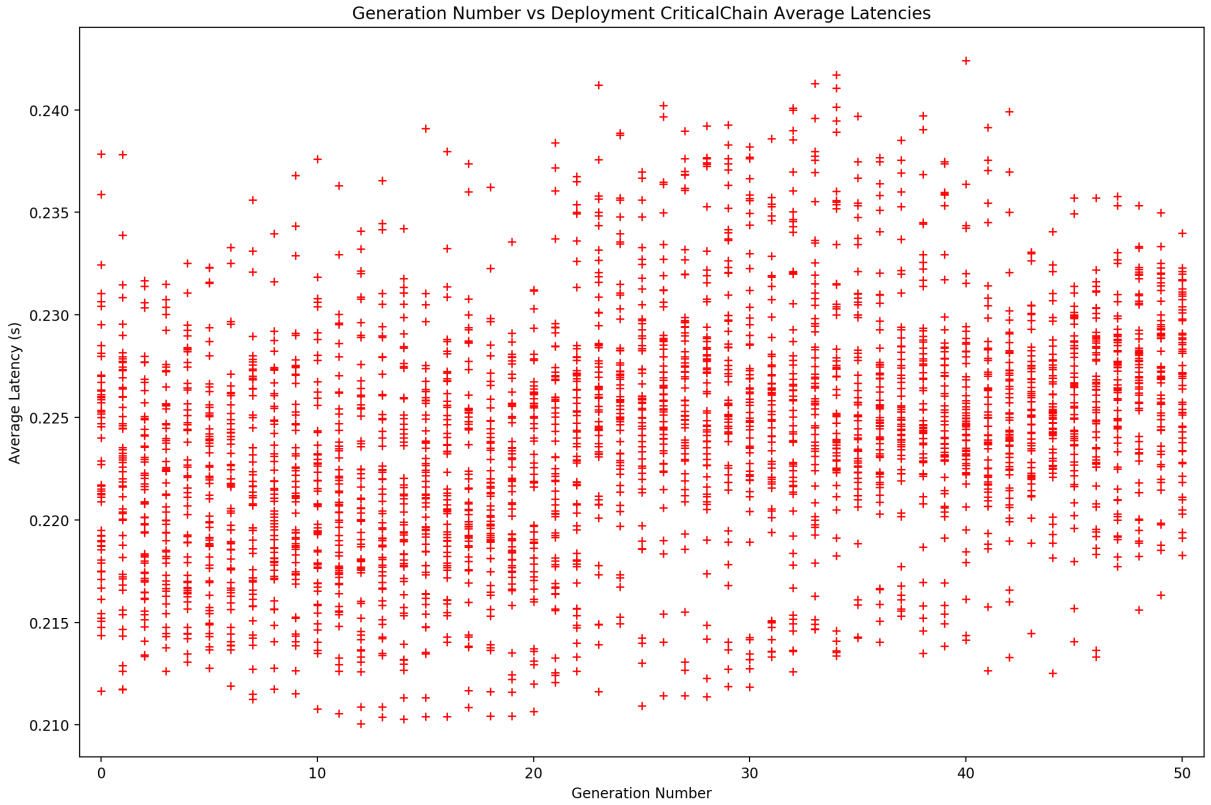

Figure 7.11: Average critical chain latency (from 58 samples) for an undersea sensor software system across 50 generations and populations of 60

Figure 7.12, Figure 7.13, Figure 7.14, Figure 7.15, Figure 7.16 and Figure 7.17 shows all the CPU maximum resource consumption, while Appendix E.2 shows all (fullsize) maximum CPU and memory resource consumption for each generation's population members for each node available for deployment.

The first observation that can be made from these results is that the CPU utilisation is still largely inline with the expectation that the optimisation algorithm aims to even out the distribution of workloads to minimise impact from large CPU utilisation delays on the latency performance of the critical chain[6]. Across the the six computing nodes, the measured workloads were distributed relatively evenly across four of the nodes (Cranberry01:5-15%, Cranberry02:15-20%, Mandarin02:5-15%, Mandarin03:5-10%), while the other two nodes (Cranberry03:20-35%, Mandarin01:30-45%) had increased levels of utilisation.

In line with observations made during the optimisation algorithm testing section, memory utilisation was basically even across the generations for each node. The variation across the generations for all the nodes ranged from approximately 12% to 18%, while each node final generation utilisation ranged from approximately 13% to 17%. As a result, there is negligible room for adjustment to improve consumption levels.

Noting that these case studies do not consider memory consumption as part of the design investigation, and therefore play no part in the optimisation process, memory utilisation for subsequent case studies was not considered.

[6]because of the variables sizes of workloads being distributed, the level of evenness for distribution will not match that being reached with equal-sized workload distributions, as seen in the previous testing

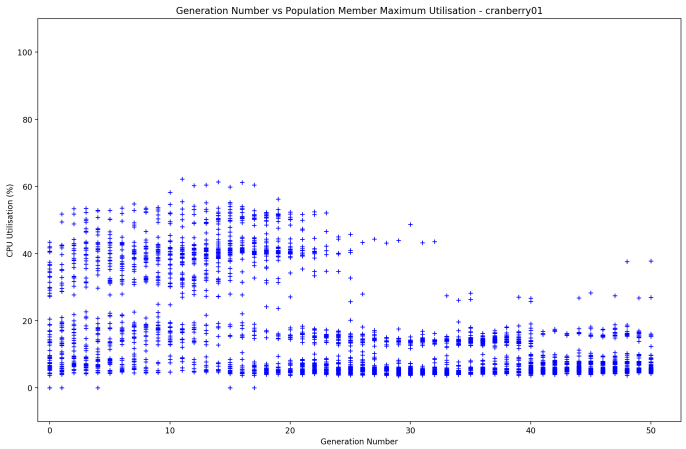

Figure 7.12: Cranberry01 maximum CPU resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

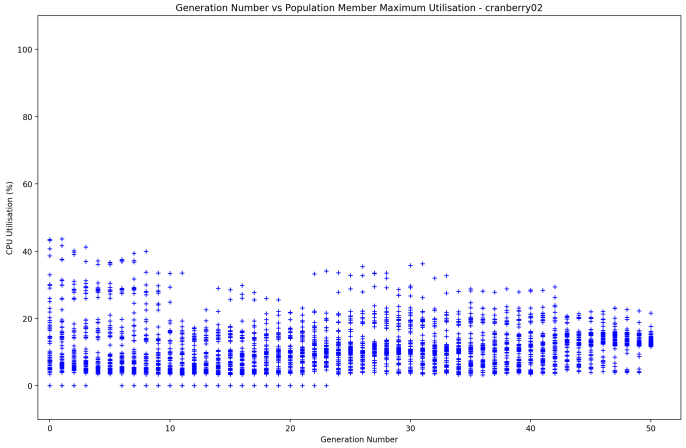

Figure 7.13: Cranberry02 maximum CPU resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

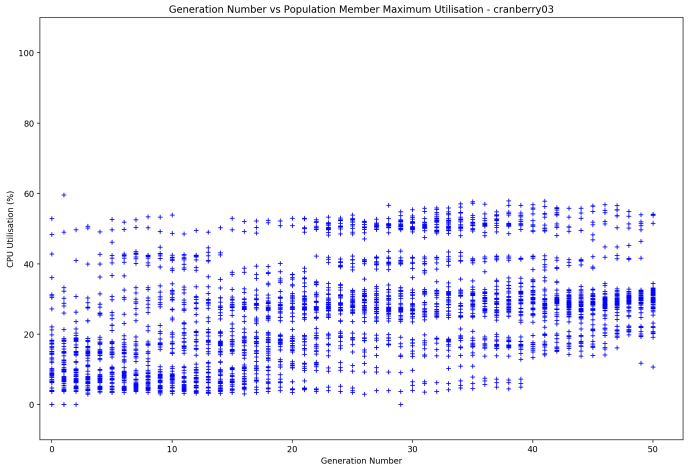

Figure 7.14: Cranberry03 maximum CPU resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

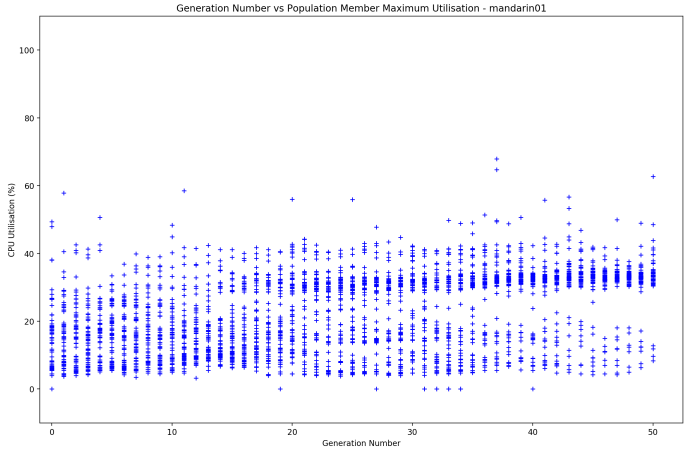

Figure 7.15: Mandarin01 maximum CPU resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

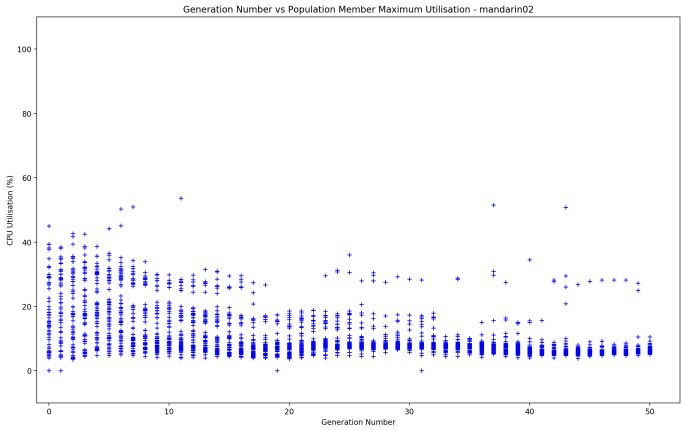

Figure 7.16: Mandarin02 maximum CPU resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

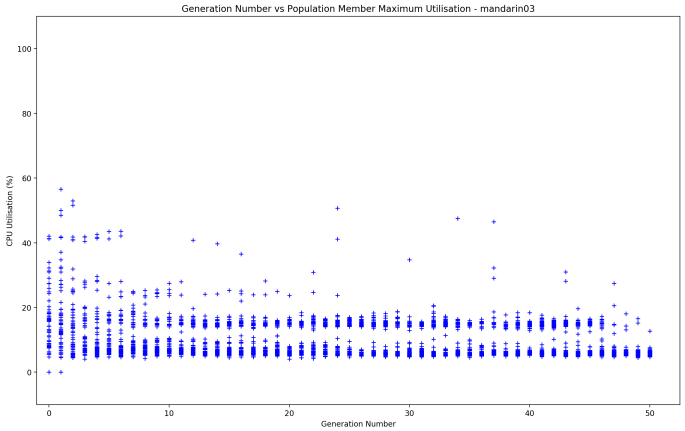

Figure 7.17: Mandarin03 maximum CPU resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

The final generation of component deployment arrays for the baseline case study model are shown in Appendix E.1, along with their population number and objective score. From this Table it can be seen that the optimisation process has converged on 41 unique deployment profiles (Appendix E.2 across five different objective scores. From the five different objective scores, the best performing objective score led to nine different deployment options capable of delivering the best performing critical chain latency performance (Table 7.4).

Table 7.4: Baseline design scenario unique deployment options

Number	Unique Component Deployment Array	Objective Score
1	[1, 2, 4, 5, 2, 4, 0, 2, 2, 2, 3, 5, 2, 2, 3, 2, 2, 3, 3]	349.53
2	[1, 2, 4, 5, 2, 4, 0, 2, 3, 2, 0, 5, 2, 3, 2, 2, 2, 2, 3]	349.53
3	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 0, 5, 2, 3, 2, 2, 2, 2, 3]	349.53
4	[1, 2, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	349.53
5	[1, 2, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 3, 2, 3, 3, 2, 3, 3]	349.53
6	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 2]	349.53
7	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	349.53
8	[1, 3, 4, 5, 3, 4, 0, 3, 2, 2, 2, 5, 2, 3, 2, 2, 2, 3, 3]	349.53
9	[1, 3, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 3, 2, 3, 2, 2, 3, 3]	349.53

As detailed in Chapter 5, the component deployment array size is based on the number of component instances present within the software system, where

each array index represents a particular component instance and the number within the cell represents the computing node to which the software component is to be deployed to. In this case study model, the component deployment array consists of 19 cells and the node numbers to be assigned range from zero to five. Table 7.5 and Table 7.6 detail which software component is assigned to which component deployment array index and what node number is assigned to what computing node within the run-time environment.

Array Index	MEDEA ID Number	Component Instance Name
0	911	PrimarySensor
1	386	TrackTx
2	376	PrimarySampleProcessing
3	1485	PrimarySampleTx
4	1865	TrackCreation
5	1157	PrimaryUnderWaterComms
6	1222	ReceiverStation
7	1277	AuxcillaryunderWaterComms
8	1802	TrackUnderWaterComms
9	1866	AuxcillaryAirComms
10	1928	TrackAirComms
11	1994	PrimaryAirComms
12	2125	SecondarySensor
13	2191	SecondarySampleProcessing
14	2253	SecondarySampleTx
15	2326	WaterSensor_1
16	2392	WaterCreation_1
17	2454	WaterTrackTx_1
18	2696	WaterSensor_2

Table 7.5: Case study deployment array software component assignment

Node Number	Computing Node Name	Undersea System Node Name
0	Cranberry01	Surface Buoy 1
1	Cranberry02	Seawater Sensor 2
2	Cranberry03	Seawater Sensor 3
3	Mandarin01	Seabed Sensor 4
4	Mandarin02	Seabed Sensor 5
5	Mandarin03	Land Receiver Station 6

Table 7.6: Case study computing node assignment

7.2 Scenario #1: Dedicated Deployment and System Failure Resilience

The first design scenario demonstrates the need to satisfy two known mandated software component deployment constraints, as well as the need to ensure a system redundancy requirement is satisfied.

The first of the known mandated deployment constraints was the need for an identified Seabed Sensor node to have a software component instance, associated with a hypothetically environmental sensor hardware to be deployed together. The second mandated deployment requirement was the need for the 'Land Receiver Station' node to only have the software component instance associated with the receiver station function alone (the *ReceiverStation*) deployed onto it. For this specialised mandated deployment constraint, where only a single software component instance can be deployed onto the receiver station node, a specialised mandated

deployment constraint was introduced within the optimisation algorithm.

For the system redundancy requirement two software component instances *PrimarySampleTx* and *SecondarySampleTx* were modelled using the *noColocation* constraint, so they would never be deployed onto the same node. This scenario represents a design change to reduce the risk of zero sample messages being transmitted upon a single computing node failure.

The specifications for this case study experiment are detailed in Table 7.8.

Configurable Attribute	**Specification**
Spare CPU Utilisation	Seawater Sensor 2 - 0% Seawater Sensor 3 - 0% Seabed Sensor 4 - 0% Seabed Sensor 5 - 0% Surface Buoy 1 - 0% Land Receiver Station 6 - 0%
Tx/Rx Bandwidth	Seawater Sensor 2 - 100Mb Seawater Sensor 3 - 100Mb Seabed Sensor 4 - 10Mb Seabed Sensor 5 - 10Mb Surface Buoy 1 - 1Gb Land Receiver Station 6 - 1Gb
Mandate Software Placement	'SecondarySensor' placed on Seabed Sensor 5 'Receiver' only placed on Land Receiver Station
No-Colocation Deployment Requirement	'PrimarySampleTx' and 'SecondarySampleTx'
Non-functional Design Change	2 & 4
Population Size	60
Generation Count	50

Table 7.7: Design Scenario 1 experimentation conditions

7.2.1 Experimentation Results

Figure 7.18 shows the objective scores converging towards a minimum across the 50 generations. It can be seen that the optimisation process has converged more quickly than the baseline experiment, where less variation in objective scores occurred around the mid-20 generation point. From the final generation of predicted results (Appendix E.4), a convergence to a single lowest objective score was found, with 37 different optimised deployment options identified (Table 7.8).

Furthermore, Table 7.8 shows the satisfaction of all three mandated deployment requirements. Deployment array index six (*ReceiverStation*) component instance) has node five (Land Receiver Station node) always allocated and index 12 (*SecondarySensor*) placed on node 4 (Seabed Sensor). While deployment array index 3 (*PrimarySampleTx*) component instance and 14 (*SecondarySampleTx*) component instance never have the same computing node allocated.

Figure 7.19 shows the average temporal performance results across the 50 generations for the critical chain identified. As a result of the introduced mandated deployment requirements, an increased number of deployment options for the best performing critical chain latency performance was found, and an improvement convergence performance was found compared to the baseline scenario. In this case, the improved convergence has also resulted in a small, improved latency performance across the identified deployment profiles[7].

[7] there is a y-axis scale change between the two graphs, with this graph being only 4ms compared with 30ms for the baseline scenario

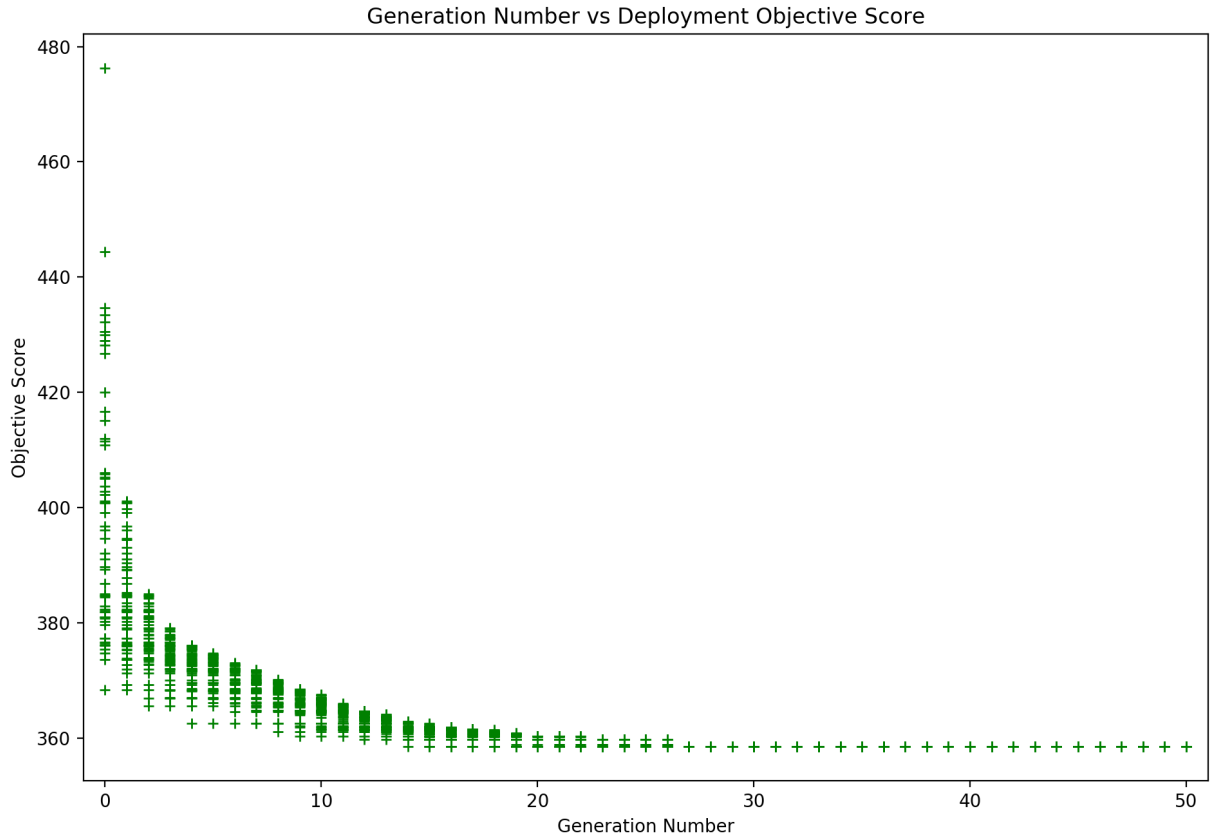

Figure 7.18: Design scenario 1 objective scores for an undersea sensor software system across 50 generations

Table 7.8: Design scenario 1 unique deployment options

Number	Component Deployment Array	Objective Score
1	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
2	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
3	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.60
4	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.60
5	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.60
6	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 2, 4, 4, 1, 2, 1, 1, 1, 2]	358.60
7	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
8	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 2, 2]	358.60
9	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
10	[0, 1, 4, 0, 1, 3, 5, 2, 2, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.60
11	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
12	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.60
13	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 2, 4, 4, 1, 2, 1, 1, 1, 1]	358.60
14	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 2, 4, 4, 2, 1, 1, 1, 2, 2]	358.60
15	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 2, 4, 4, 2, 1, 2, 1, 1, 1]	358.60
16	[0, 1, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
17	[0, 1, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 2, 2, 1, 1, 1]	358.60
18	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
19	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 2, 1]	358.60

Continued on next page

Table 7.8 – continued from previous page

Number	Component Deployment Array	Objective Score
20	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
21	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.60
22	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.60
23	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
24	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.60
25	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.60
26	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.60
27	[0, 2, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
28	[0, 2, 4, 0, 2, 3, 5, 1, 1, 1, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
29	[0, 2, 4, 0, 2, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
30	[0, 2, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
31	[0, 2, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
32	[0, 2, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.60
33	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
34	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
35	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.60
36	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.60
37	[0, 2, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60

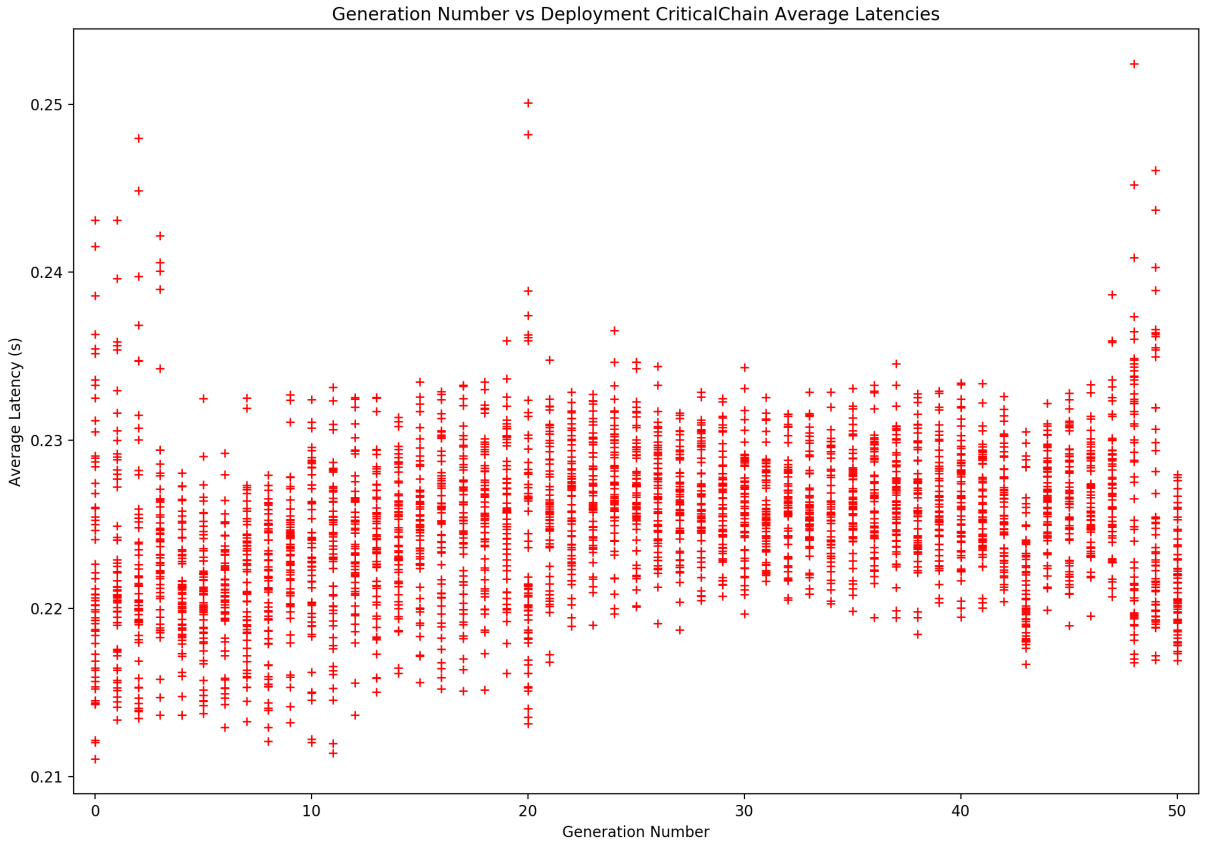

Figure 7.19: Average critical chain latency for an undersea sensor software system with mandated deployment requirements (50 generations with populations of 60, and 58 samples for each population member)

Figures 7.20, 7.21, 7.22, 7.23, 7.24 and 7.25 shows the maximum CPU resource consumption for generation members of each node available for deployment with this design scenario experiment. The first observation that can be made is that CPU utilisation on Mandarin03 is reasonably consistent across the generation, which is expected as this node only has the single software component instance (*ReceiverStation*) deployed to it.

It can also be seen that a largely even distribution of workload is occurring across three nodes (Cranberry01:15-18%, Mandarin01:5-10% and Mandarin02:10-15%), while the remaining two nodes (Cranberry02: 15-50% and Cranberry03:5-55%) have a similarly increased level of utilisation compared with the baseline case study experiment. Once again, expected behaviour is found due to the optimisation algorithm looking to minimise the impact of CPU resource utilisation delays and impact on the critical chain latency performance.

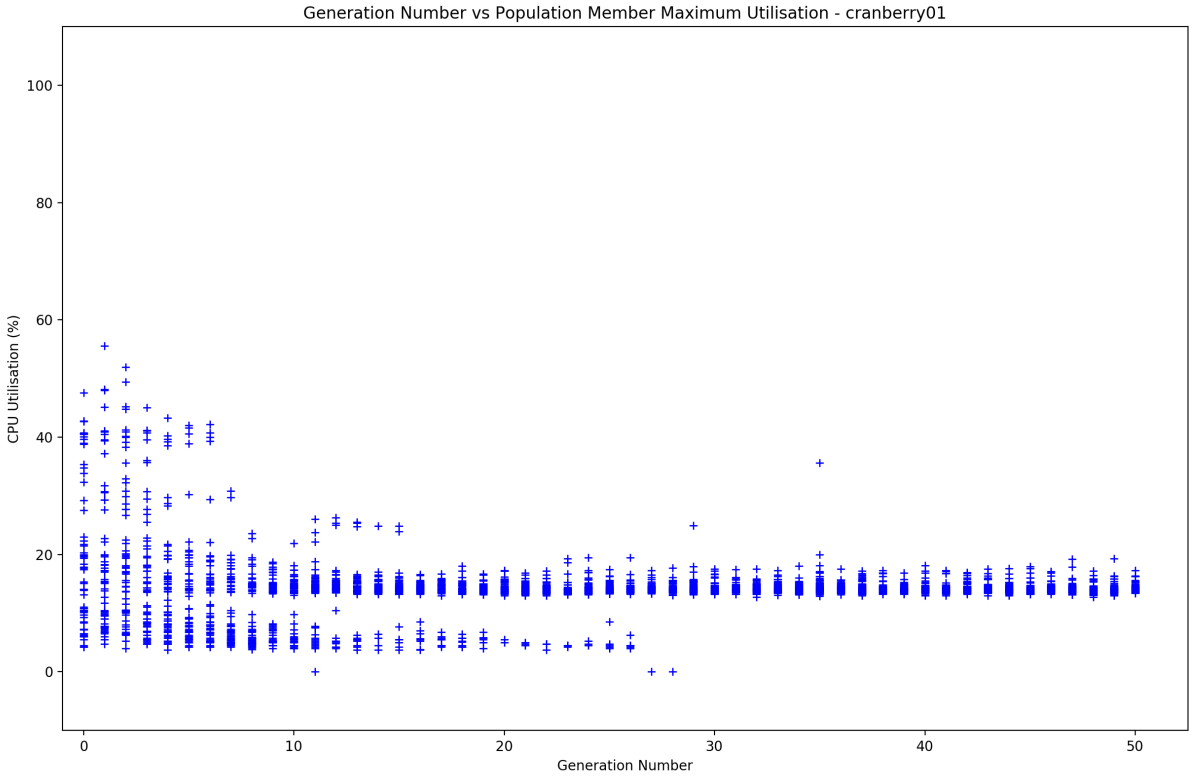

Figure 7.20: Cranberry01 maximum CPU resource consumption for an undersea sensor software system for the design scenario 1 experiment (50 generations with populations of 60, and 58 samples for each population member)

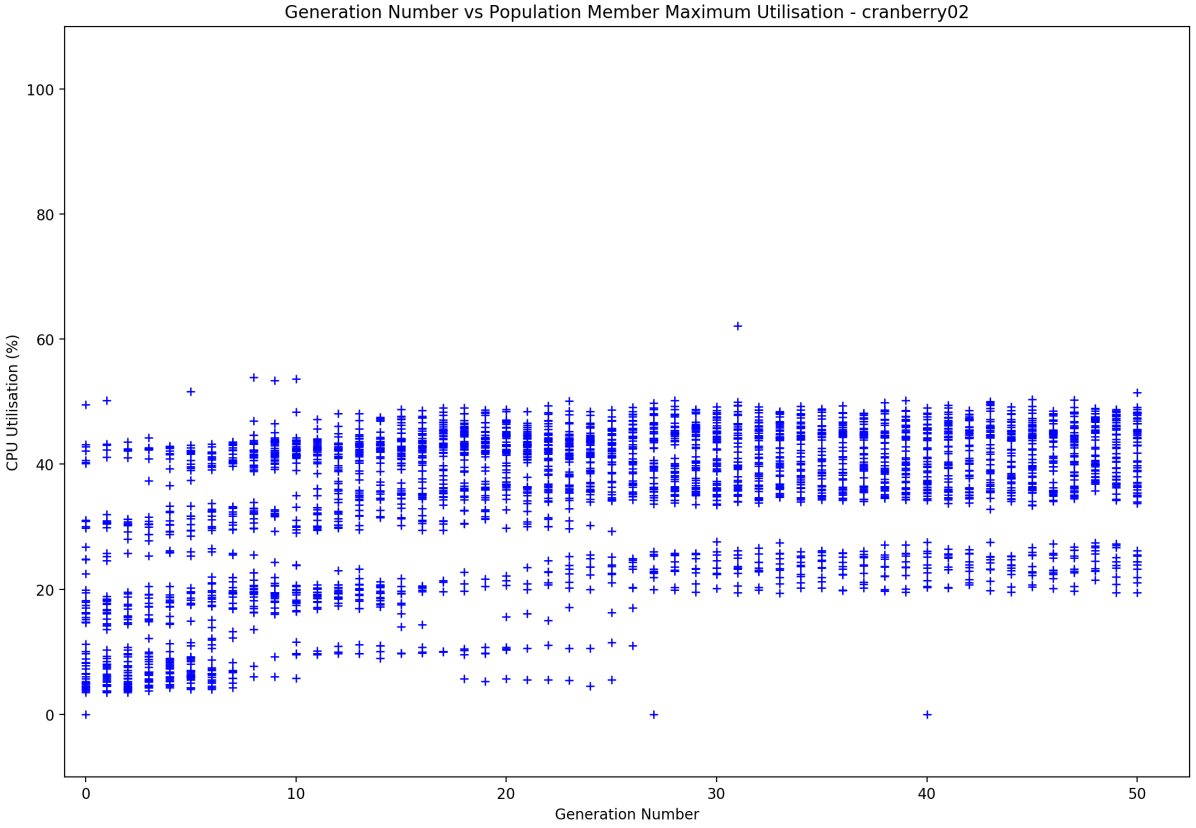

Figure 7.21: Cranberry02 maximum CPU resource consumption for an undersea sensor software system for the design scenario 1 experiment (50 generations with populations of 60, and 58 samples for each population member)

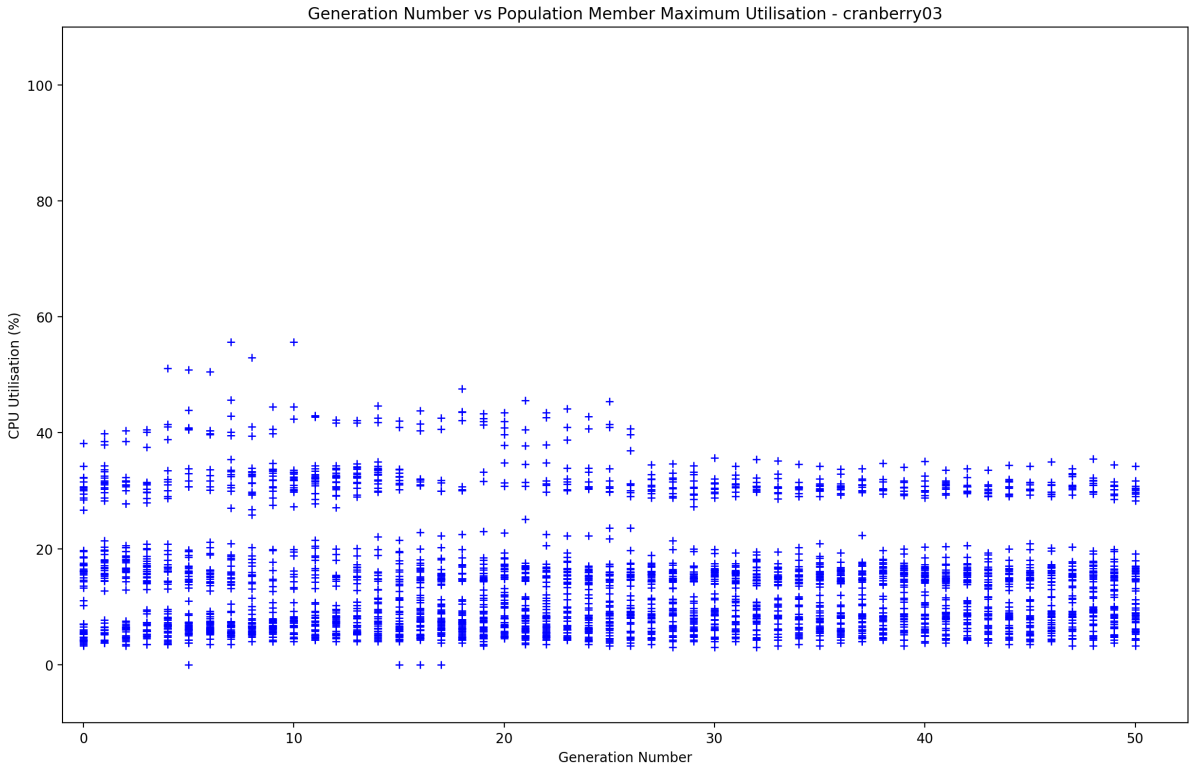

Figure 7.22: Cranberry03 maximum CPU resource consumption for an undersea sensor software system for the design scenario 1 experiment (50 generations with populations of 60, and 58 samples for each population member)

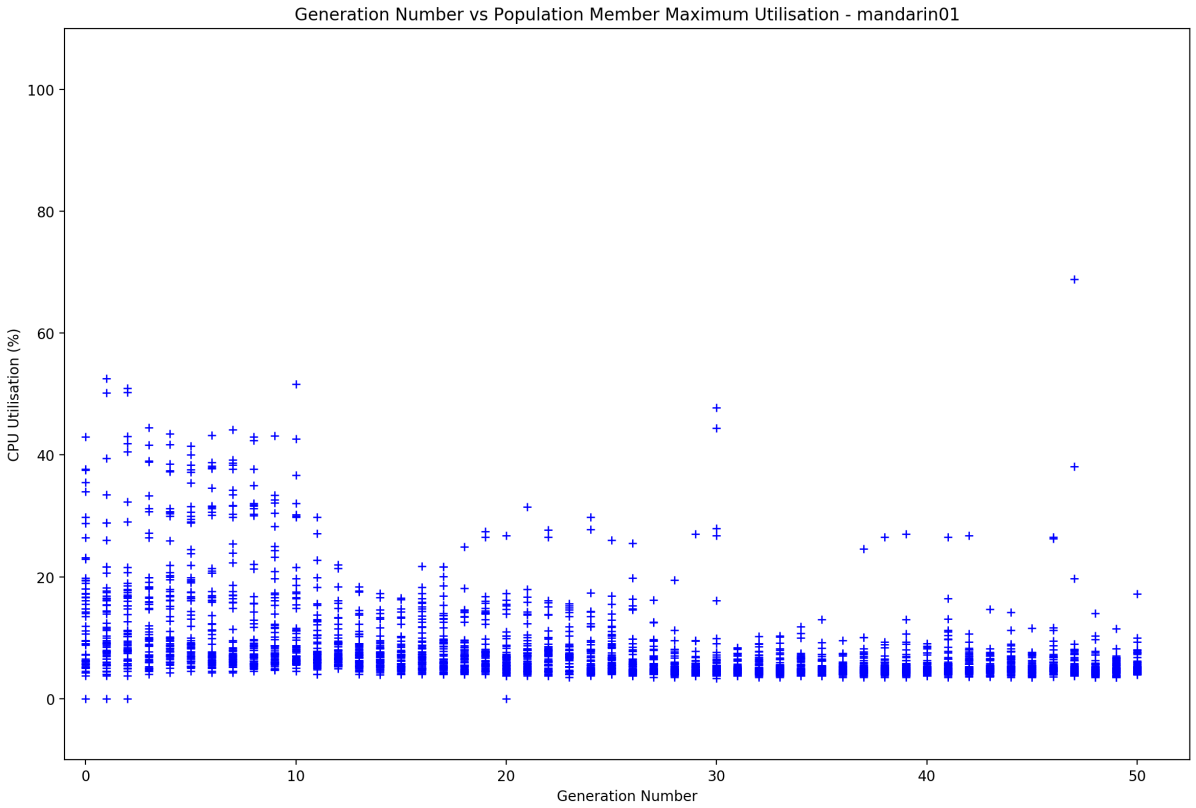

Figure 7.23: Mandarin01 maximum CPU resource consumption for an undersea sensor software system for the design scenario 1 experiment (50 generations with populations of 60, and 58 samples for each population member)

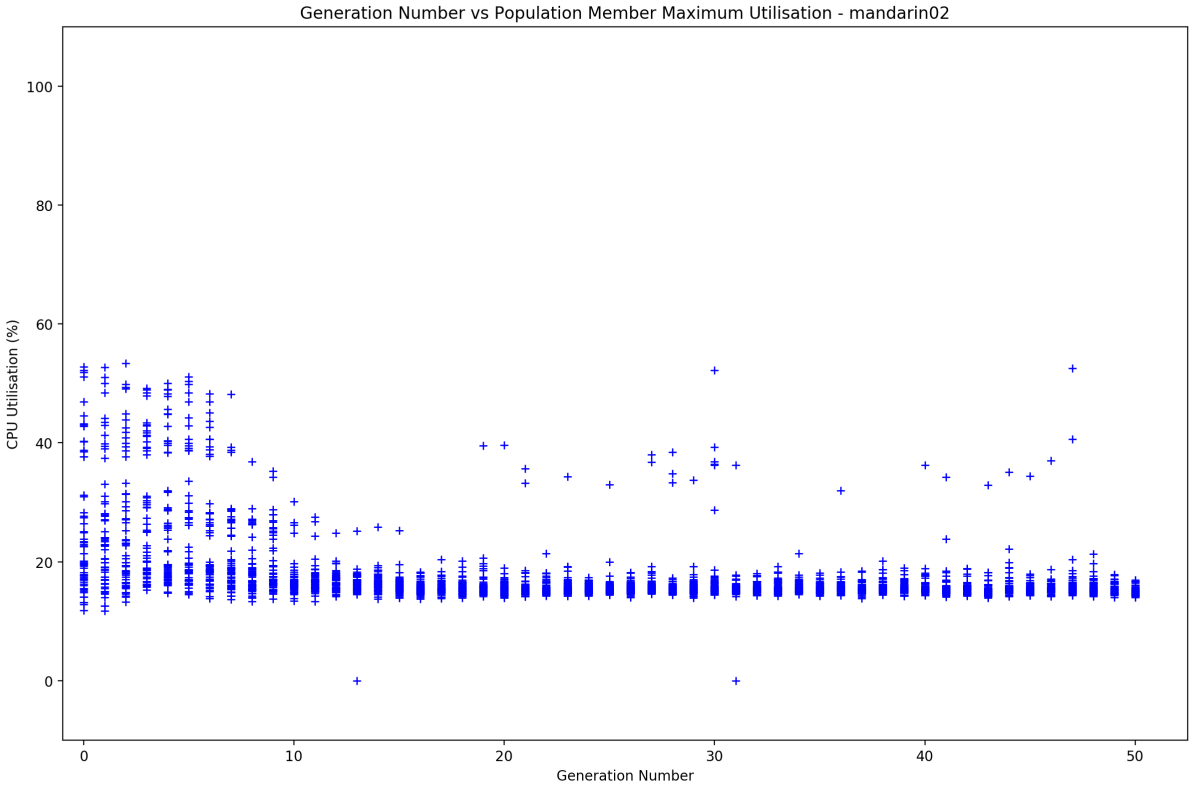

Figure 7.24: Mandarin02 maximum CPU resource consumption for an undersea sensor software system for the design scenario 1 experiment (50 generations with populations of 60, and 58 samples for each population member)

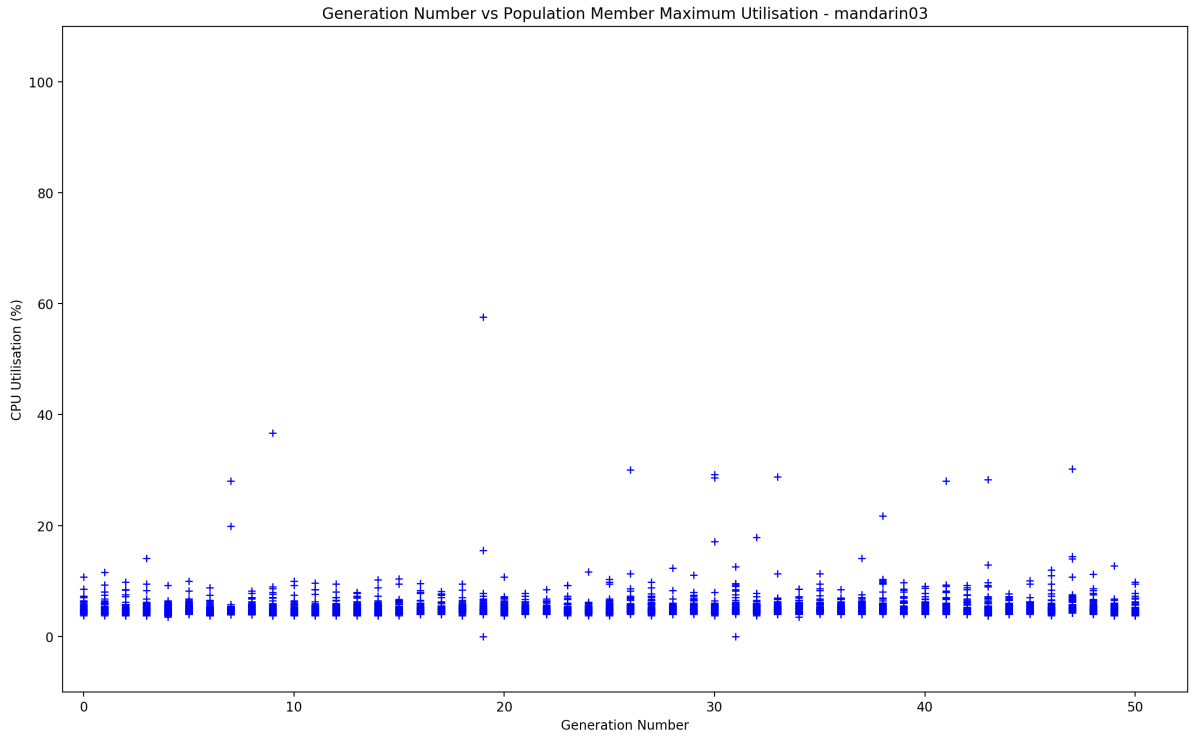

Figure 7.25: Mandarin03 maximum CPU resource consumption for an undersea sensor software system for the design scenario 1 experiment (50 generations with populations of 60, and 58 samples for each population member)

7.3 Scenario #2: Undersea Sensor Network Power Budget Reductions

Building on the opening system design scenario investigation, this scenario adds an operational need to improve power utilisation and explore how to increase time between battery replacements (and hypothetically reduce the resourcing associated with such a task) for some nodes. As a result, a lower consumption of battery power from the available nodes was required and achieved through changes in spare CPU utilisation for those nodes.

The specifications for this design scenario experiment are detailed in Table 7.9.

7.3.1 Experimentation Results

Appendix E.5 shows the final generation population members for this design scenario. The first observation made was that the objective scores are less than those achieved in the previous design scenarios. This indicates the optimisation process has identified component deployment options that are producing resource utilisation that is less than the defined constraint. As a result, the objective scores are further reduced, based on the gap between the defined threshold and predicted utilisation. We can also see that the optimisation algorithm has produced the final set of population members across only three, different, minimised objective scores (Appendix E.6) and identified a single deployment option with the best objective score (Table 7.10).

Configurable Attribute	Specification
Spare CPU Utilisation	Seawater Sensor 2 - 70% Seawater Sensor 3 - 70% Seabed Sensor 4 - 40% Seabed Sensor 5 - 40% Surface Buoy 1 - 30% Land Receiver Station 6 - 30%
Tx/Rx Bandwidth	Seawater Sensor 2 - 100Mb Seawater Sensor 3 - 100Mb Seabed Sensor 4 - 10Mb Seabed Sensor 5 - 10Mb Surface Buoy 1 - 1Gb Land Receiver Station 6 - 1Gb
Mandate Software Placement	'SecondarySensor' placed on Seabed Sensor 5 'Receiver' only placed on Land Receiver Station
No-Colocation Deployment Requirement	'PrimarySampleTx' and 'SecondarySampleTx'
Non-functional Design Change	1, 2 & 4
Population Size	60
Generation Count	50

Table 7.9: Design Scenario 2 experimentation conditions

Table 7.10: Design scenario 2 final generation unique deployment options details

Number	**Component Deployment Array**	**Objective Score**
1	[4, 4, 0, 3, 4, 1, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	304.75

Inline with the resource utilisation testing conducted in Section 6 and the overall modelling philosophies of exploring worst case performance scenarios, Figures 7.26, 7.27, 7.28, 7.29, 7.30 and 7.31 shows the largest maximum CPU resource consumption across the population members for each generation for each node available for deployment for this design scenario experiment.

From these plots, we see the optimisation search results identified non-conventional component deployment options, and possibly a result that would unlikely be found through any manual effort (assuming a manual approach is possible and practical). We can observe that the optimisation algorithm had identified solutions where CPU utilisation has been reduced below the threshold for five out of the six available nodes. In some cases the resultant CPU resource consumption has been reduced significantly (50-75% below the defined constraint).

More specifically, the optimisation has resulted in CPU resource consumption close to the required thresholds for 'Cranberry02' and 'Cranberry03', while 'Mandarin01' sees a consumption sitting around 25% below the required threshold. In the case of 'Cranberry01' and 'Mandarin03' the consumption are significantly reduced and sit below the threshold at around the 75% and 50% accordingly.

While these results show impressive outcomes in reducing the CPU resource utilisation compared with what is required, the impact of this was the optimisation algorithm chose to over-utilise the CPU resources on 'Mandarin02'. In fact, the utilisation was beyond the 80% hard threshold. This resulted from the fact that the under-utilisation objective scores improvements (reductions) out weighed the objective score increases from over-utilisation. As a result, the overall objectives score was reduced and identified outcomes produced.

Figure 7.32 shows the maximum CPU utilisation across the population members for each generation for Mandarin02. From this graph we can see there is an outlier maximum CPU utilisation result responsible for going beyond the 80% hard threshold point, but the majority of the population member CPU utilisation results

are still in the over-utilisation realm.

While such a result is a correct outcome and may be acceptable, it would be reasonable to assume that in most cases this would not be a desirable outcome from the system designer aspect. In these situations, the use of the weights described in Chapter 5 would be utilised. To combat the over-utilisation on 'Mandarin02' and reduce the potential for these kinds of outcomes, the system designers can apply a weight for CPU spare utilisation for 'Mandarin02. This weight would then apply an additional influence on the optimisation algorithm and aim to guide identified solutions away from a direction that sees the over-allocation occurring.

Future research efforts would also look to introduce some improvements to the optimisation algorithm that would better consider the over-allocation impacts, such as the 80% hard threshold.

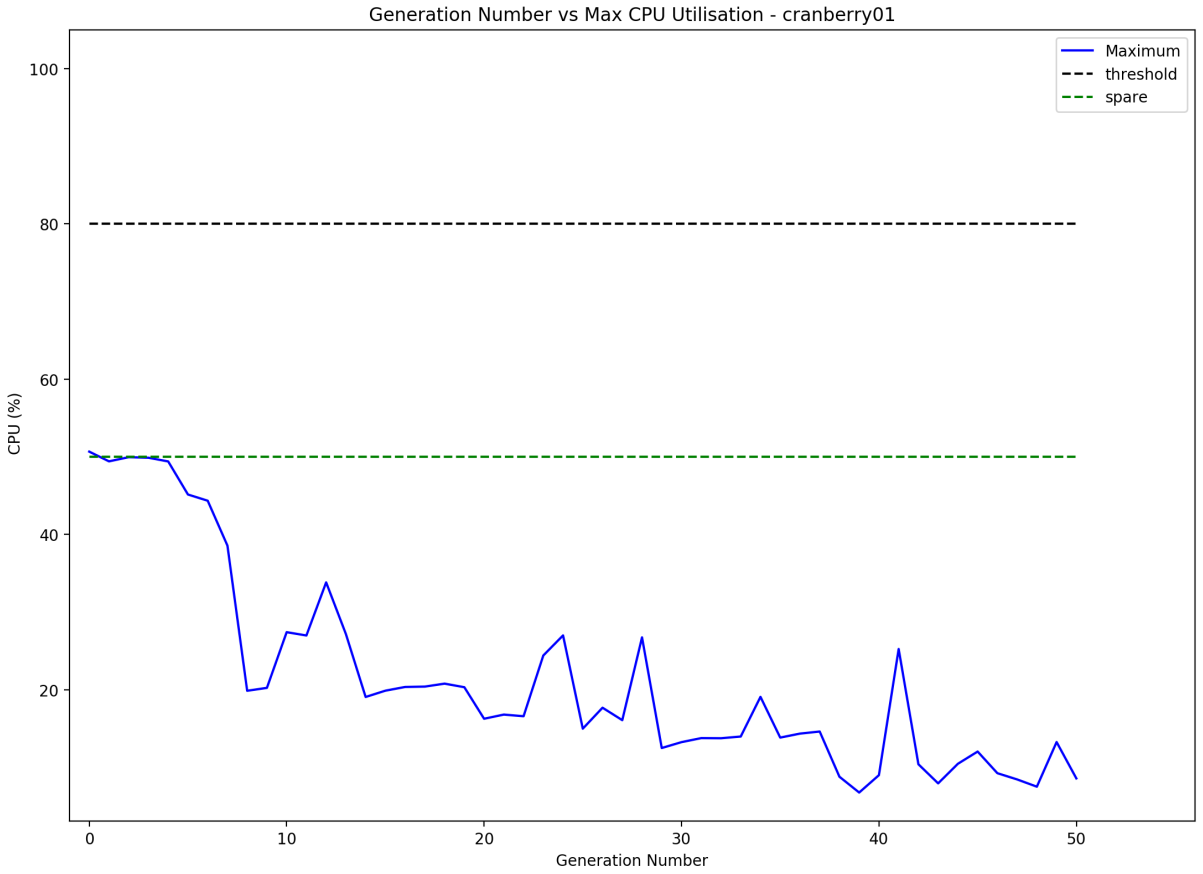

Figure 7.26: Cranberry01's largest maximum CPU resource consumption for an undersea sensor software system for the design scenario 2 experiment (50 generations with populations of 60, and 58 samples for each population member)

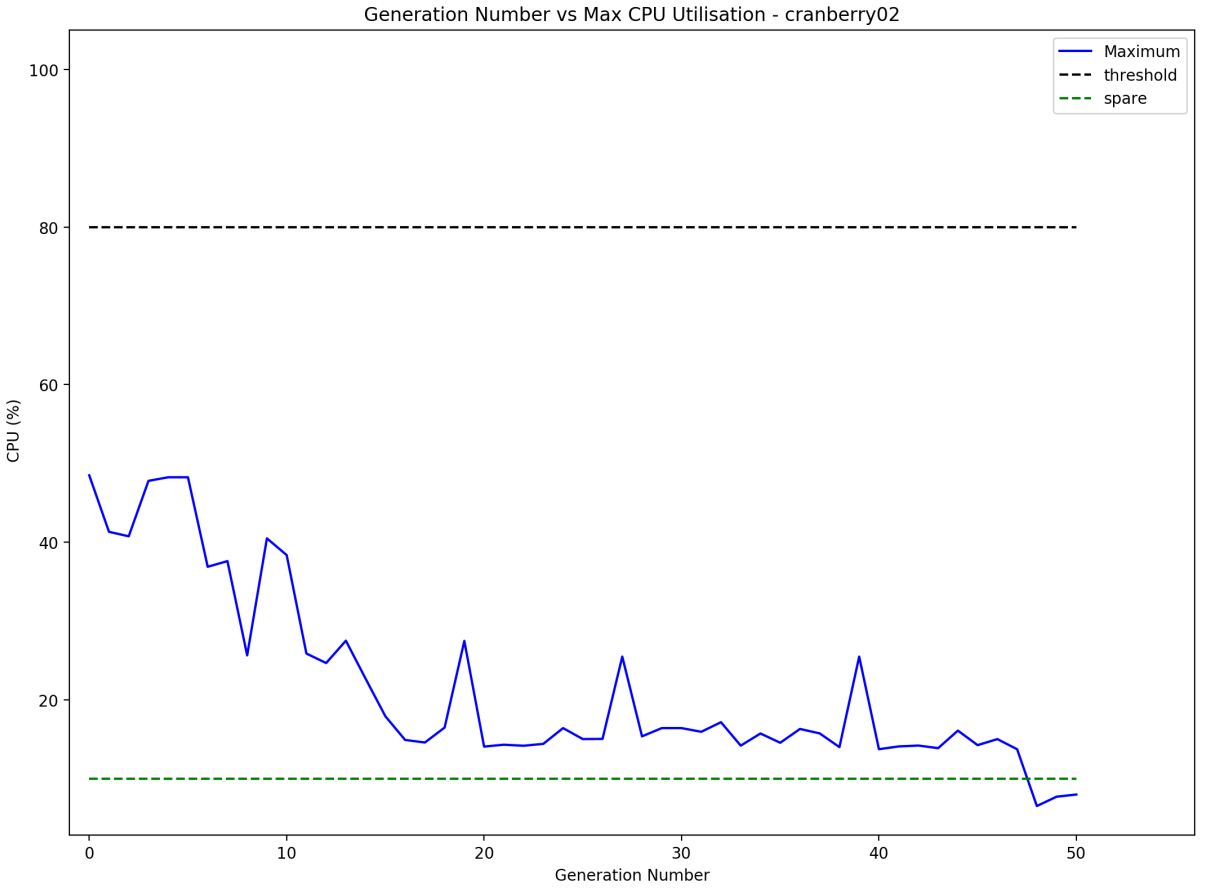

Figure 7.27: Cranberry02's CPU largest maximum resource consumption for an undersea sensor software system for the design scenario 2 experiment (50 generations with populations of 60, and 58 samples for each population member)

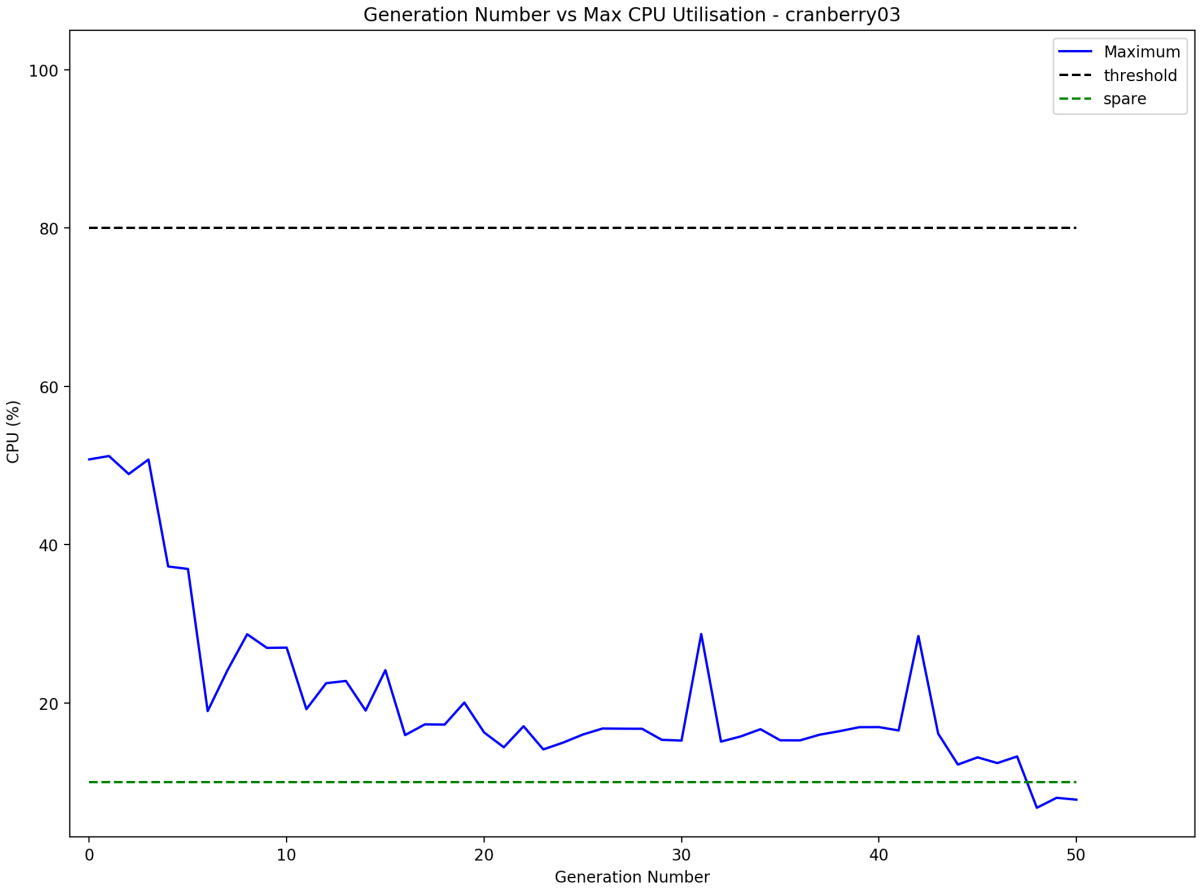

Figure 7.28: Cranberry03's CPU largest maximum resource consumption for an undersea sensor software system for the design scenario 2 experiment (50 generations with populations of 60, and 58 samples for each population member)

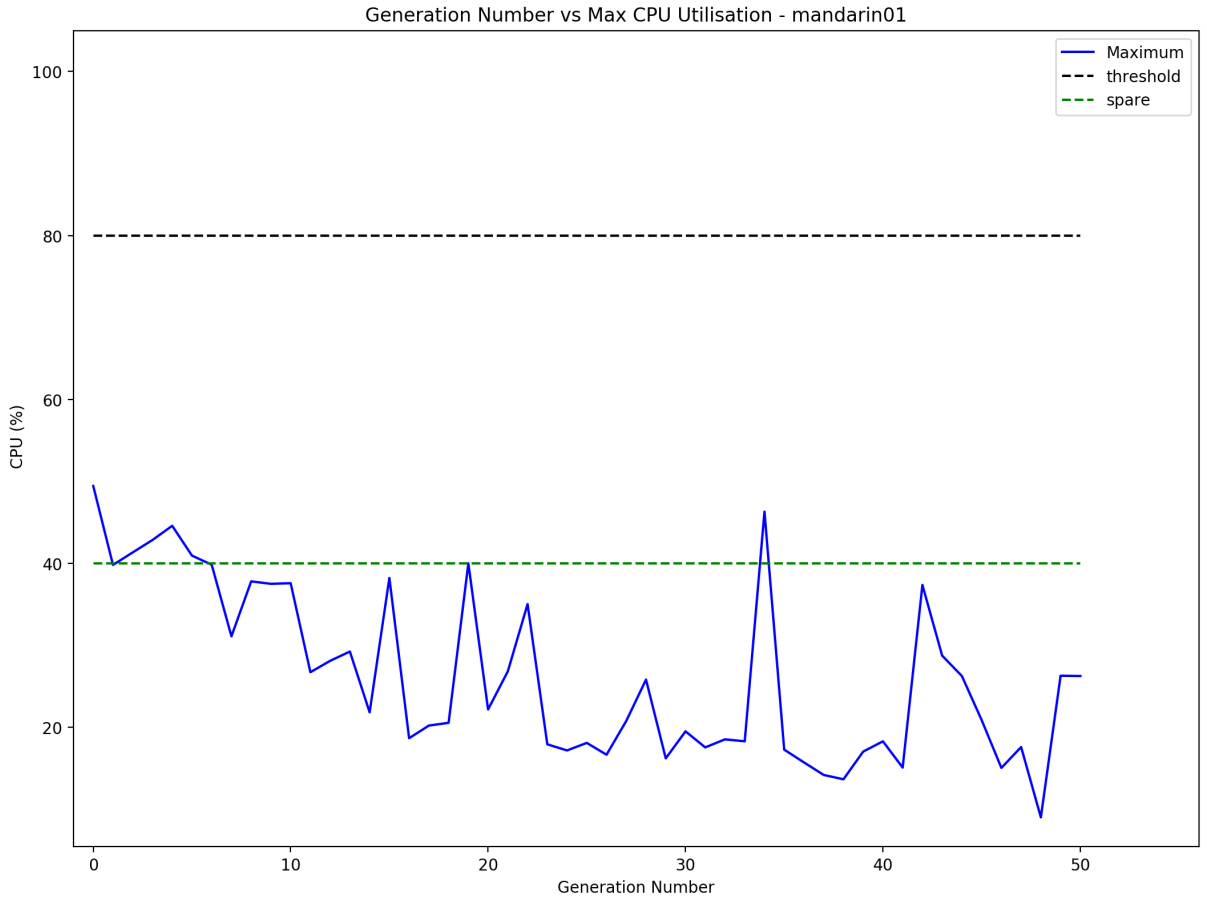

Figure 7.29: Mandarin01's CPU largest maximum resource consumption for an undersea sensor software system for the design scenario 2 experiment (50 generations with populations of 60, and 58 samples for each population member)

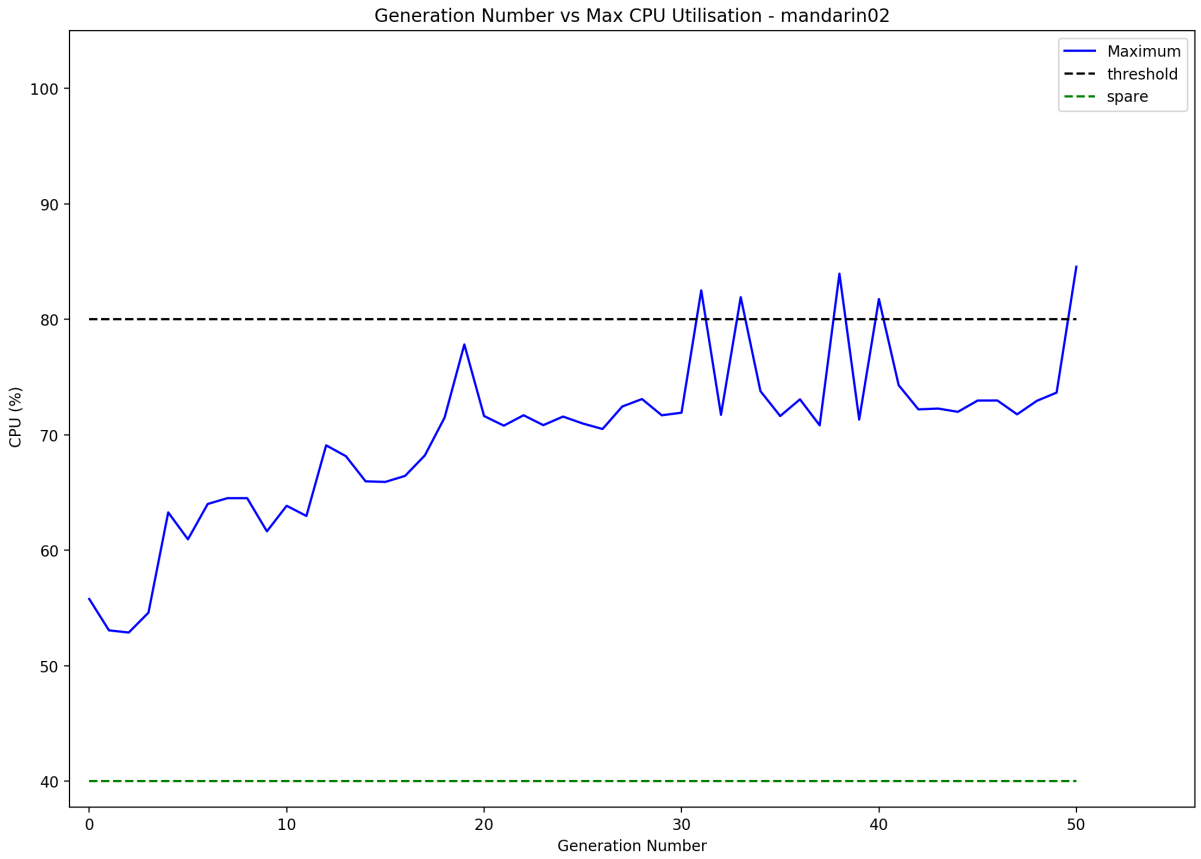

Figure 7.30: Mandarin02's CPU largest maximum resource consumption for an undersea sensor software system for the design scenario 2 experiment (50 generations with populations of 60, and 58 samples for each population member)

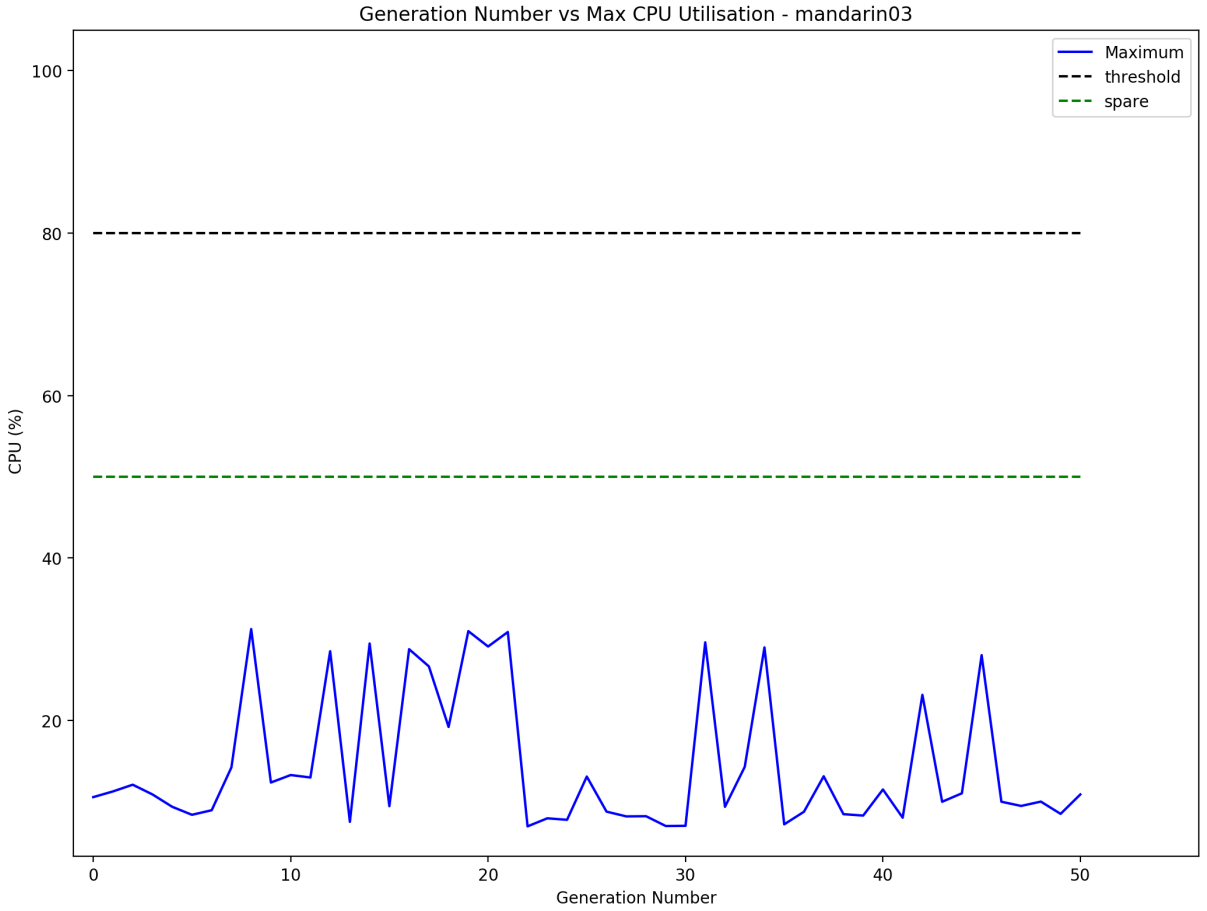

Figure 7.31: Mandarin03's CPU largest maximum resource consumption for an undersea sensor software system for the design scenario 2 experiment (50 generations with populations of 60, and 58 samples for each population member)

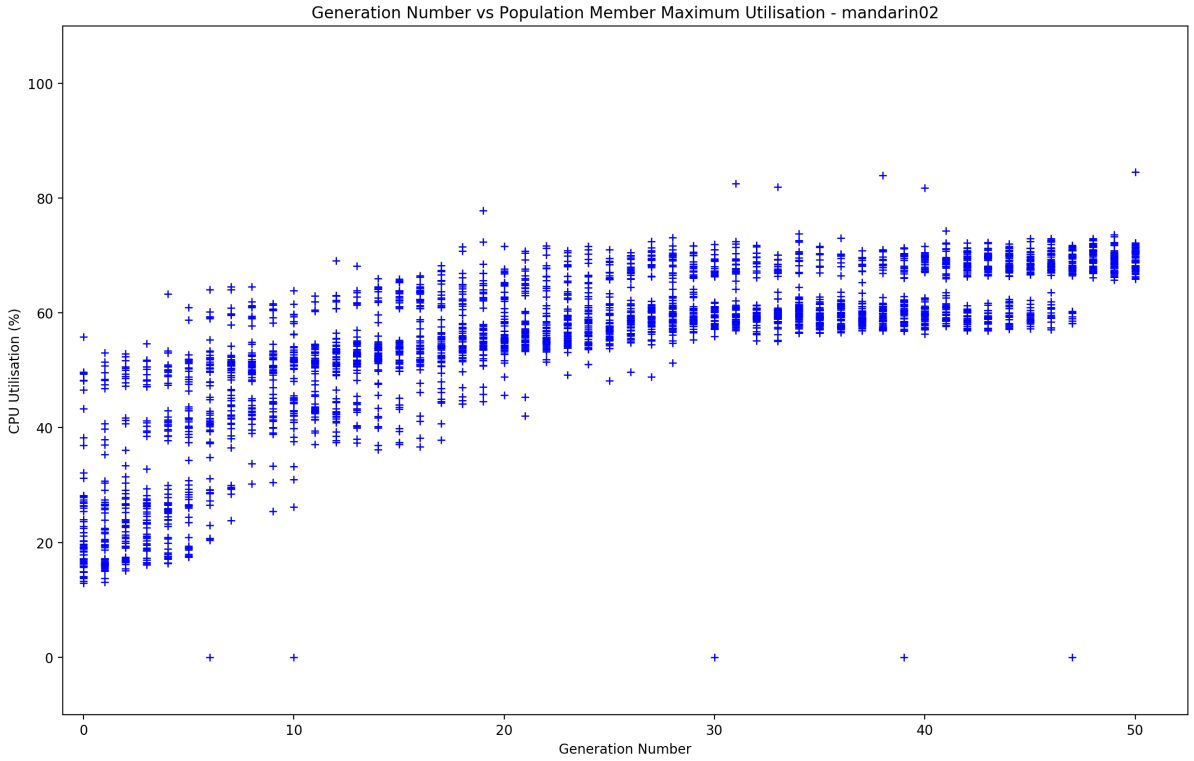

Figure 7.32: Mandarin02's CPU maximum resource consumption for population members for each generation for the design scenario 2 experiment (50 generations with populations of 60, and 58 samples for each population member)

Lastly Figure 7.33 and Figure 7.34 show the objective scores and critical chain latencies across the generations. For the objective score results found in Figure 7.33, the optimisation algorithm behaviour and performance for reaching the final generation objective score were different from the previous design scenarios. Based on the increased number of constraints and requirements, the convergence performance is slower, while the rate of objective score reduction is also more variable.

In the case of the critical chain latency results, we see that with the introduction of the new constraints and requirements the optimisation algorithm's (and overall objective score) main influence was from CPU. As a result the final generation critical chain latencies showed both less convergence and increases in latencies.

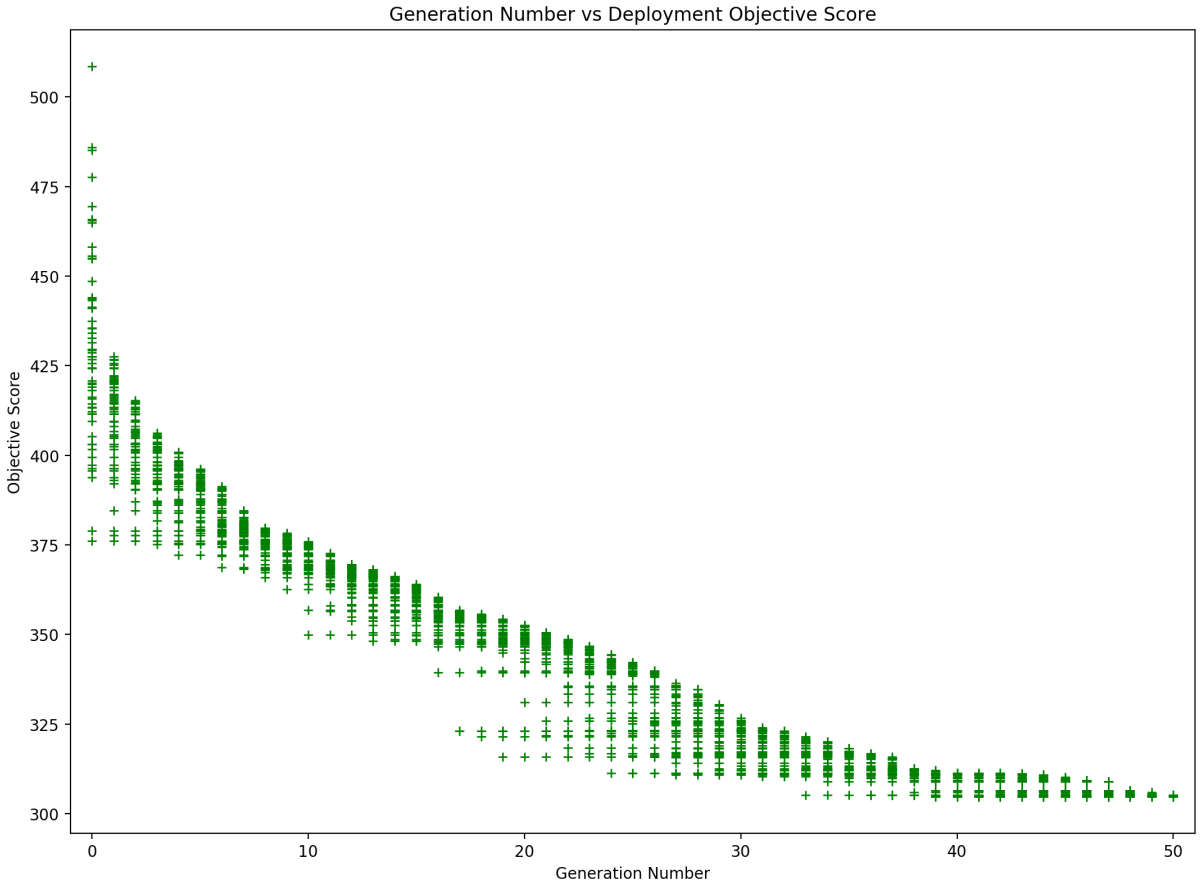

Figure 7.33: Design scenario 2 objective scores for an undersea sensor software system across 50 generations

Figure 7.34: Design scenario 2 Average critical chain latency for an undersea sensor software system (50 generations with populations of 60, and 58 samples for each population member)

7.4 Scenario #3: Improved Data Quality of Service (QoS)

The last system design scenario investigation introduced the additional design change to ensure data senescence performance for identified software component connections are being satisfied. Therefore, in addition to ensuring the identified critical chain temporal performance is being minimised as much as possible (while also considering the other defined system design constraints and requirements), the optimisation process is also considering additional temporal performance constraints for identified segments of the critical chain.

As a result, the investigation needs to explore software deployment options that consider latency performance for a string of software component connections (a segment of the critical thread), as well as a maximum latency threshold set for the connection between the two components.

This design scenario explored the software deployment profiles that best accommodate the impact of these changes. The specifications for this design scenario experiment are detailed in Table 7.11

7.4.1 Experimentation Results

Appendix E.6 shows the final generation population members for this design scenario. The first observation that can be made is that the objective scores have now

Configurable Attribute	Specification
Spare CPU Utilisation	Seawater Sensor 2 - 70% Seawater Sensor 3 - 70% Seabed Sensor 4 - 40% Seabed Sensor 5 - 40% Surface Buoy 1 - 30% Land Receiver Station 6 - 30%
Tx/Rx Bandwidth	Seawater Sensor 2 - 100Mb Seawater Sensor 3 - 100Mb Seabed Sensor 4 - 10Mb Seabed Sensor 5 - 10Mb Surface Buoy 1 - 1Gb Land Receiver Station 6 - 1Gb
Interface Max. Latency	'PrimarySensor': output port 'PrimarySampleProcessing': input port Max latency: 60ms
String Max. Latency.	'PrimarySensor': output port 'PrimarySampleProcessing': input port 'PrimarySampleTx': output port 'PrimaryUnderWaterComms': input port Max latency: 170ms
Mandate Software Placement	'SecondarySensor' placed on Seabed Sensor 5 'Receiver' only placed on Land Receiver Station
No-Colocation Deployment Requirement	'PrimarySampleTx' and 'SecondarySampleTx'
Non-functional Design Change	1, 2, 3 & 4
Population Size	60
Generation Count	50

Table 7.11: Design Scenario 3 experimentation conditions

increased considerably compared with those achieved in previous design scenarios. This indicates that while the optimisation process has identified component deployment options that consider all the requirements and constraints, the predicted performance outcomes are falling short in some areas compared with their defined

thresholds.

It can also be seen that the optimisation algorithm has produced the final set of population members across a much larger number of objective scores (20 different scores). Furthermore, across these 20 objective scores, the optimisation process identified 38 unique component deployment options (Appendix E.8), while the best performing objective identified a single deployment profile (Table 7.12).

Table 7.12: Design scenario 3 final generation unique component deployment details

Number	Component Deployment Array	Objective Score
1	[1, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	769.35

Figure 7.35 and Figure 7.36 shows the predicted interface and string latency performances across the generations for this case study, while Figure 7.37 shows the overall objective scores across the generations. As highlighted previously with the increased objective scores, both the graphed predicted results for the interface and the string latency show that the required performance thresholds are not being met at any point across the generations. The final generation predicted latency performance for the interface is approximately 66ms (when the required threshold was 60ms) and the string is approximately 197ms (when the required threshold was 170ms). Looking at the overall graphed objective scores across the generations, the performance and behaviour of the optimisation process is once again a rapid

convergence early in the generation count with very little fluctuation in the rate of reduction of the objective score beyond that knee point (approximately generation 3) of the graph, while Figure 7.38 shows the resultant critical chain latency results.

Figure 7.39, 7.40, 7.41 and 7.42 shows the maximum and average results for both the interface and the string latency measurements. From these, we can observe that the predicted results coming out of the optimisation process are pessimistic compared with what is being measured, which is the desired outcome and intended consequences of the calibration process and application of the calibration factors. While there are still differences between predicted and measured results that require further investigation and possibly algorithm adjustments with future research, in general the results show a good account of the worst-case performance scenarios.

The second observation that can be made is that the convergence performance is not as good as those found during the string and interface latency prediction testing in Chapter 6. As opposed to the previous testing, where only the latency requirements were being tested, this design scenario optimisation is trying to optimise competing requirements and constraints from mandated deployment and CPU resource utilisation definitions. As a result, the search outcomes are less ideal and consolidated, and lead to a larger spread.

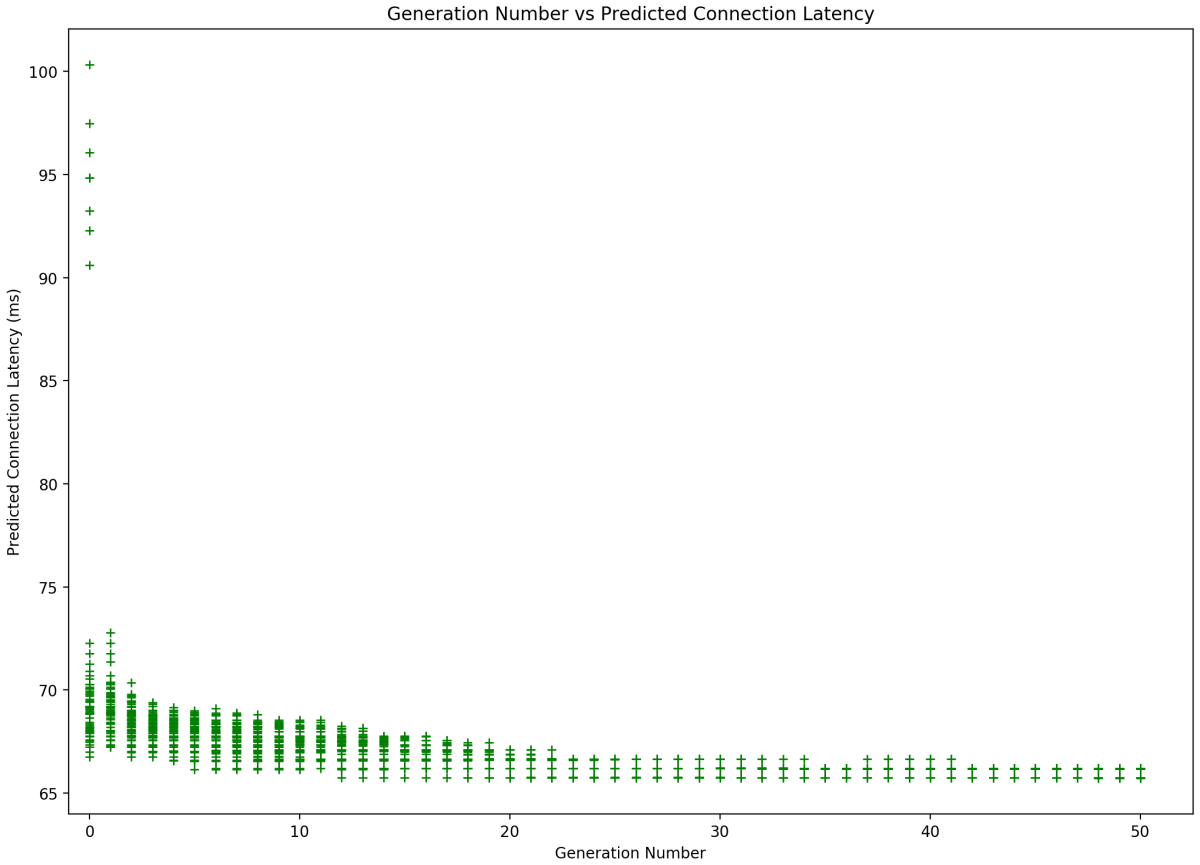

Figure 7.35: Design scenario 3 predicted interface latencies for an undersea sensor software system across 50 generations

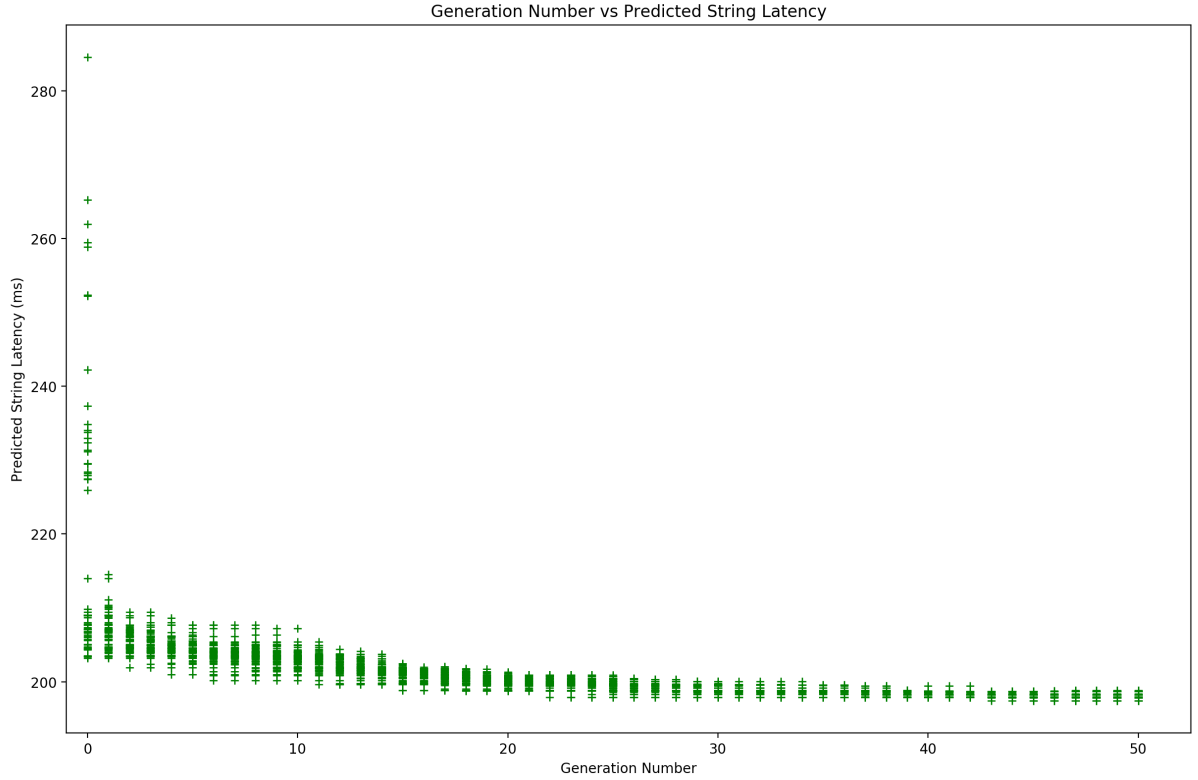

Figure 7.36: Design scenario 3 predicted string latencies for an undersea sensor software system across 50 generations

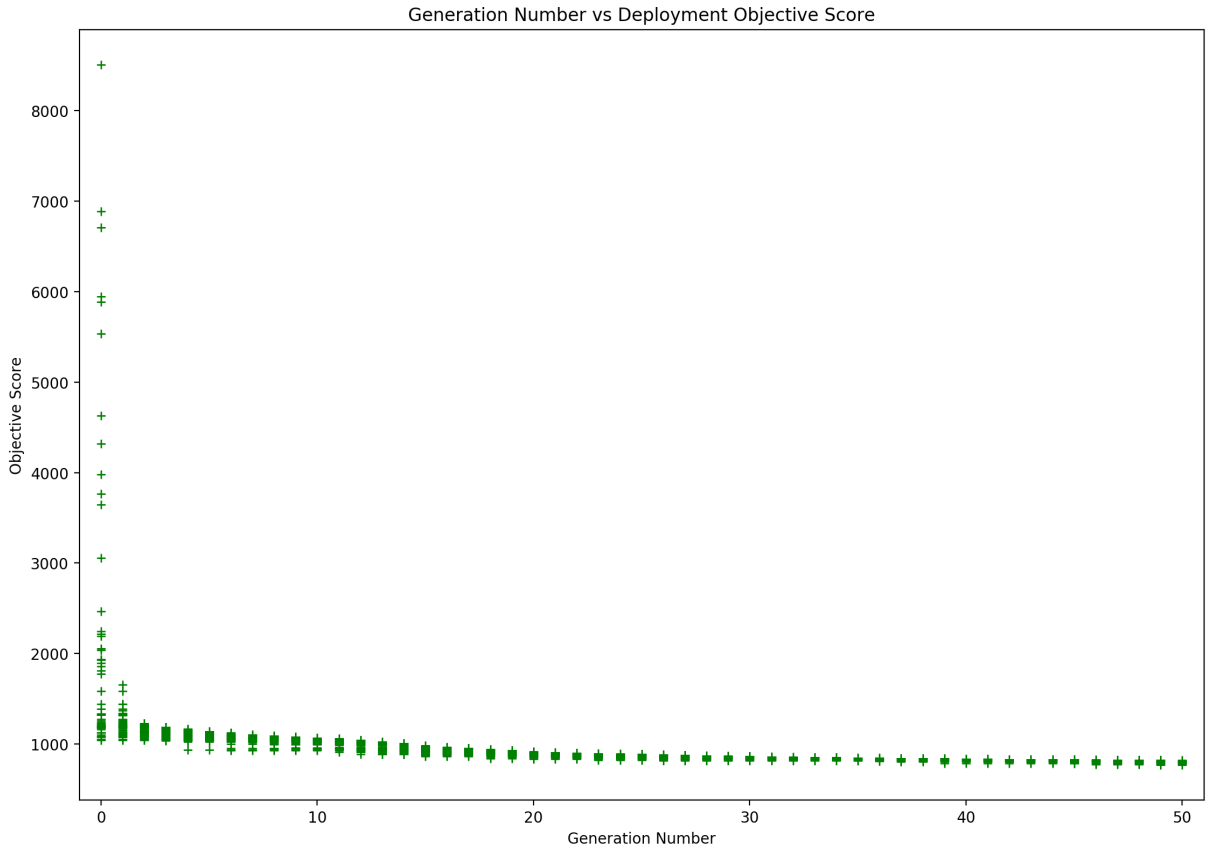

Figure 7.37: Design scenario 3 objective scores for an undersea sensor software system across 50 generations

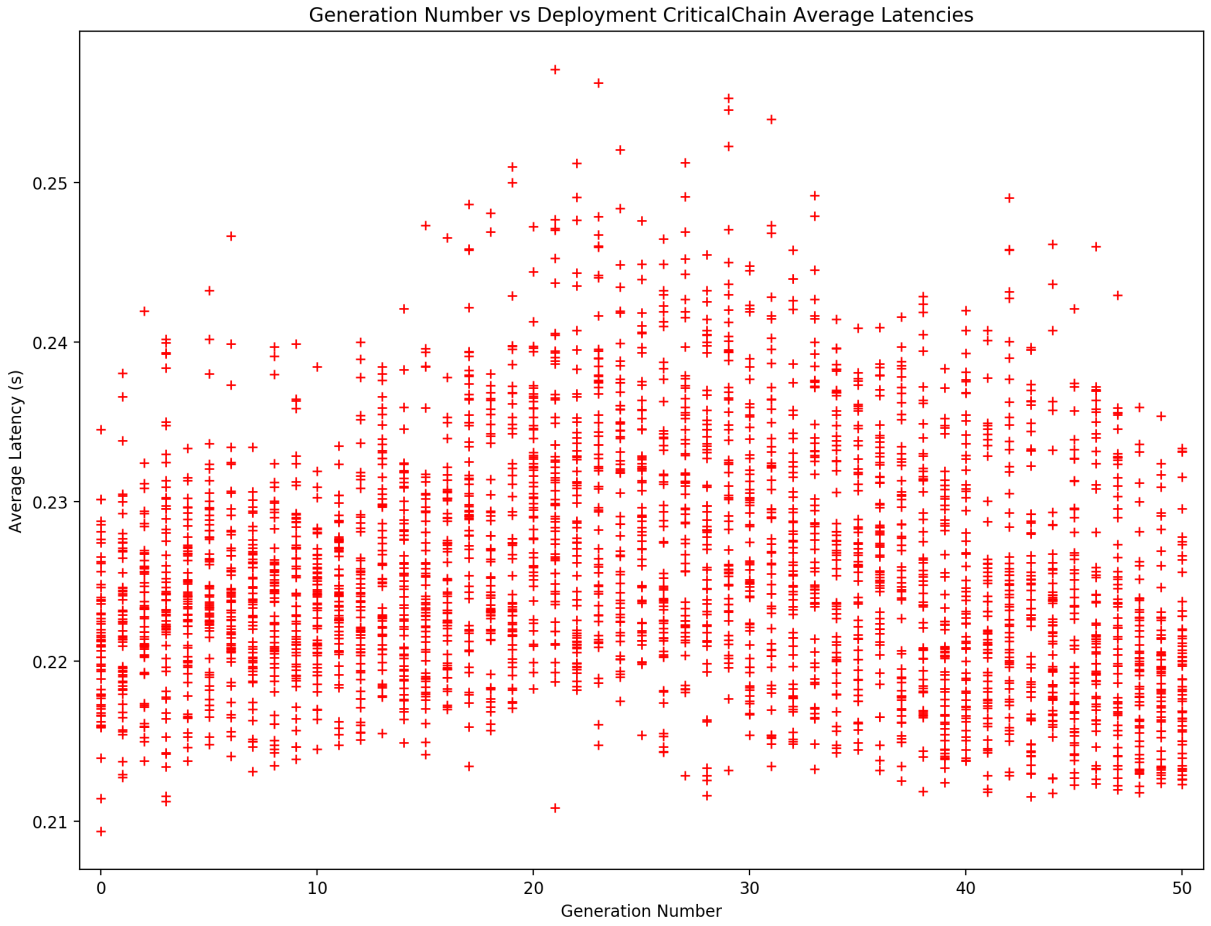

Figure 7.38: Design scenario 3: critical chain average latency for an undersea sensor software system across 50 generations (50 generations with populations of 60, and 58 samples for each population member)

The maximum interface latency results (Figure 7.39) show a convergence across the generations. In the case of the average latency results (Figure 7.40), the convergence is more pronounced. More importantly, it can be seen that the largest interface latency results and convergences drive towards the required threshold and eventually go below this at the final generation. The existence of a single outlier in the final generation can be seen just above the threshold, but in general, the measured results show an alignment with the desired optimisation behaviour and a reasonable account of the worst case latency performances. Future research will look into the small number of outliers to determine why they occur and establish methods to better account for them more effectively.

As with the interface latency results, the string latency results (Figure 7.41 and Figure 7.42) also show convergence across the generations, with the average results convergence once again being more pronounced. Importantly again, the convergence towards the final generation leads to the results being below the required threshold. Once more, there are a few outliers that should be investigated with future research effort, but in general, the results align with the desired optimisation behaviour and outcomes.

Figures 7.43, 7.44, 7.45, 7.46, 7.47 and 7.48 shows the largest maximum CPU resource consumption across the generations for each node available for deployment within this case study experiment.

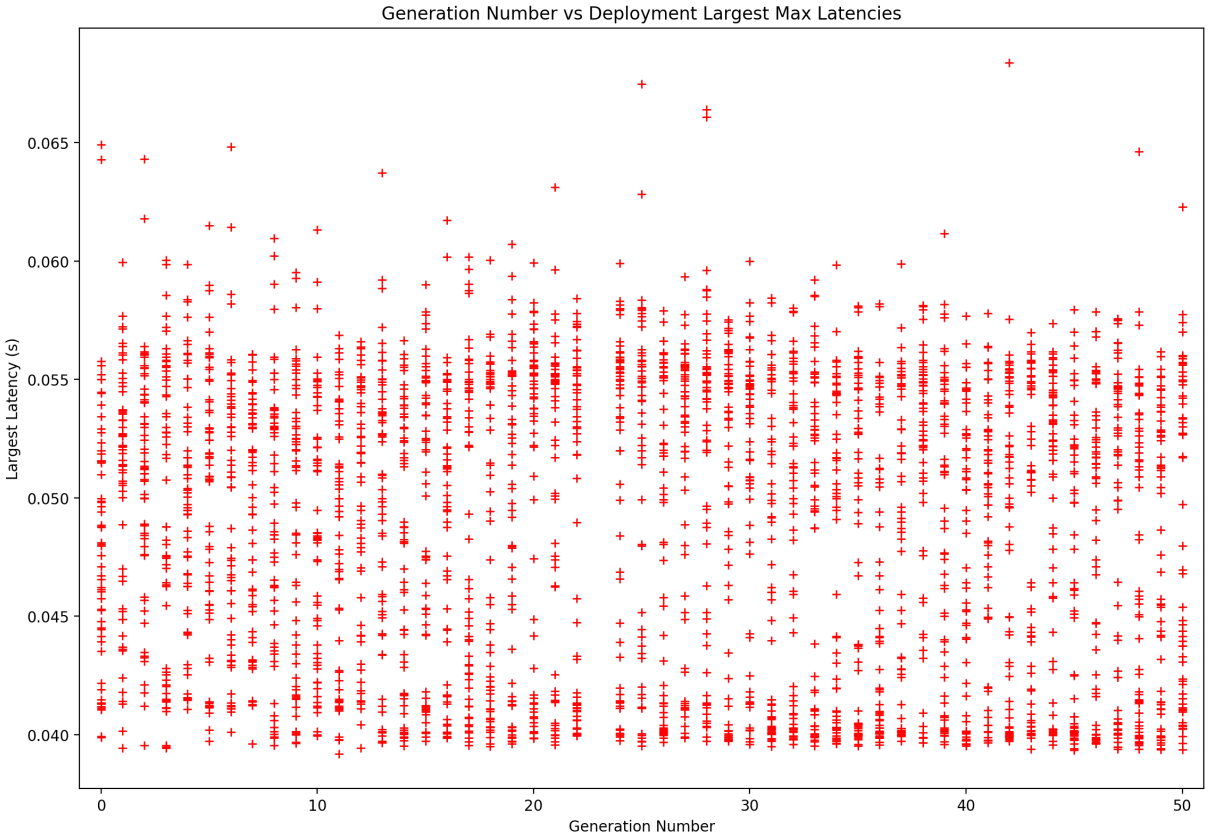

Figure 7.39: Design scenario 3: maximum measured interface latencies for an undersea sensor software system (50 generations with populations of 60, and 58 samples for each population member)

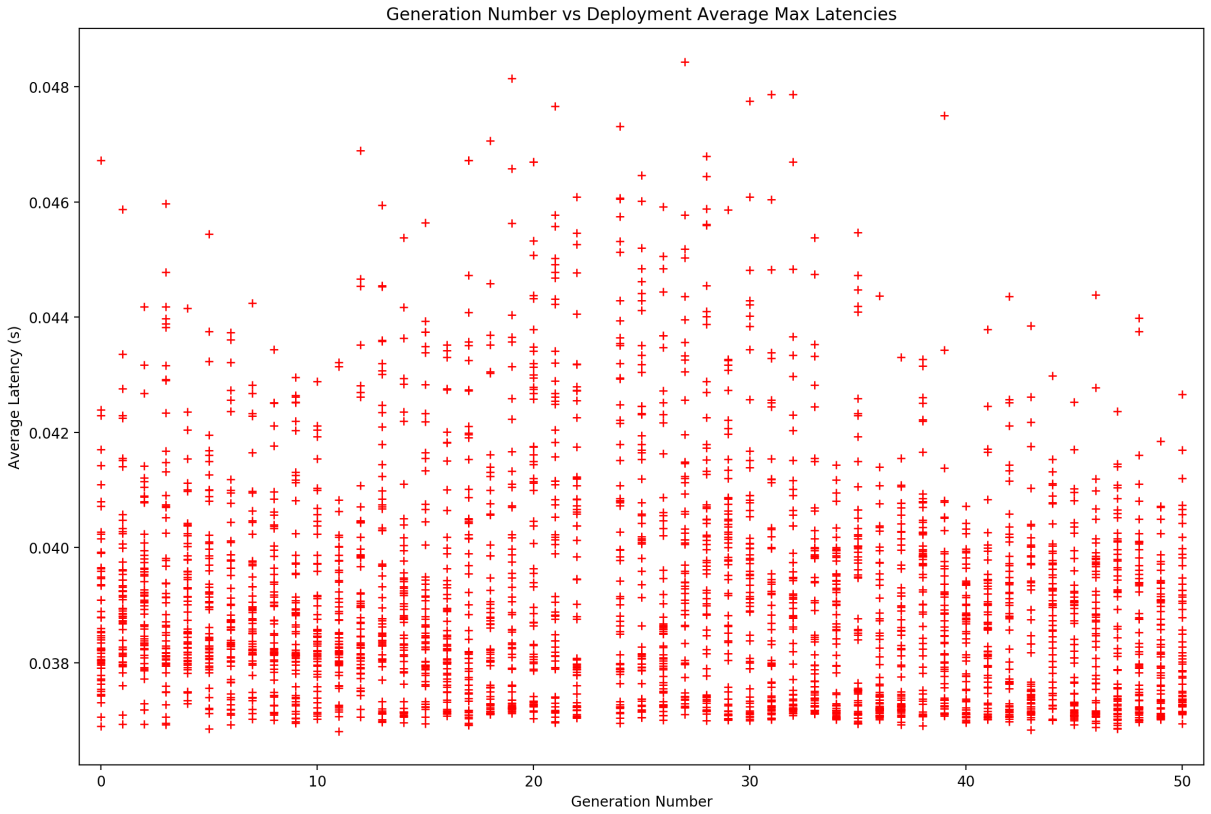

Figure 7.40: Design scenario 3: average measured interface latencies for an undersea sensor software system (50 generations with populations of 60, and 58 samples for each population member)

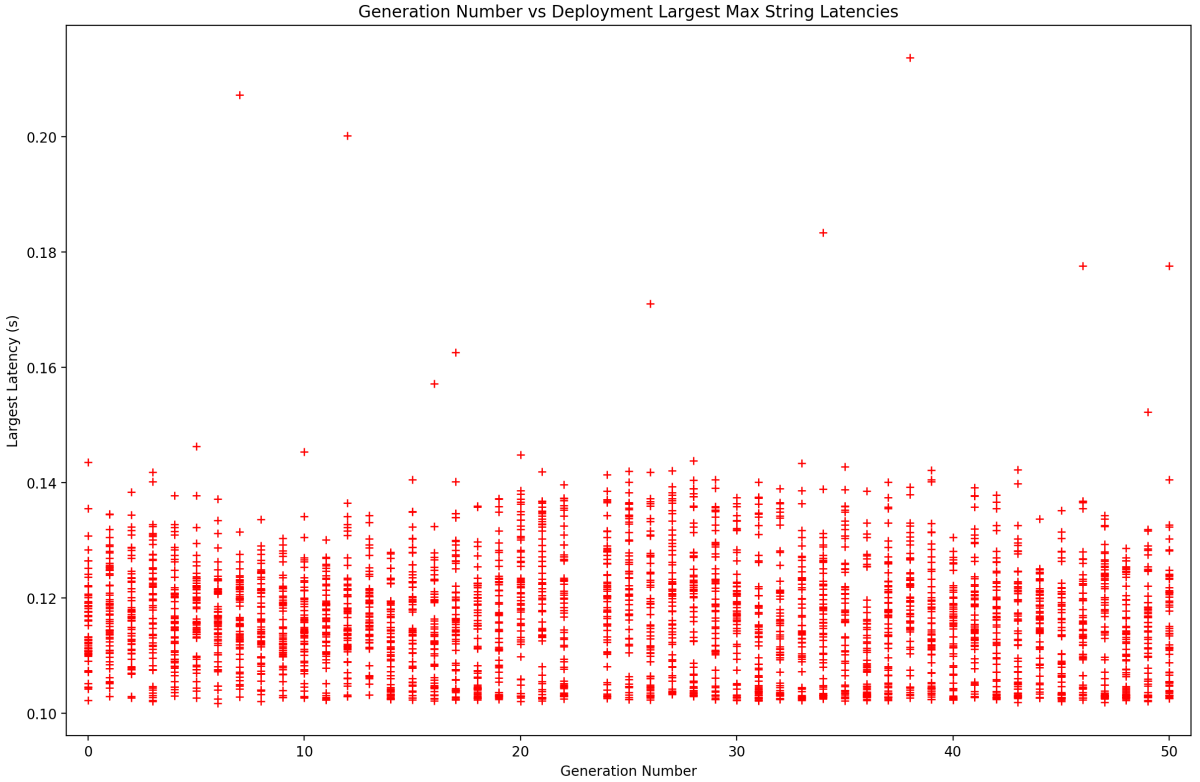

Figure 7.41: Design scenario 3: maximum measured string latencies for an undersea sensor software system (50 generations with populations of 60, and 58 samples for each population member)

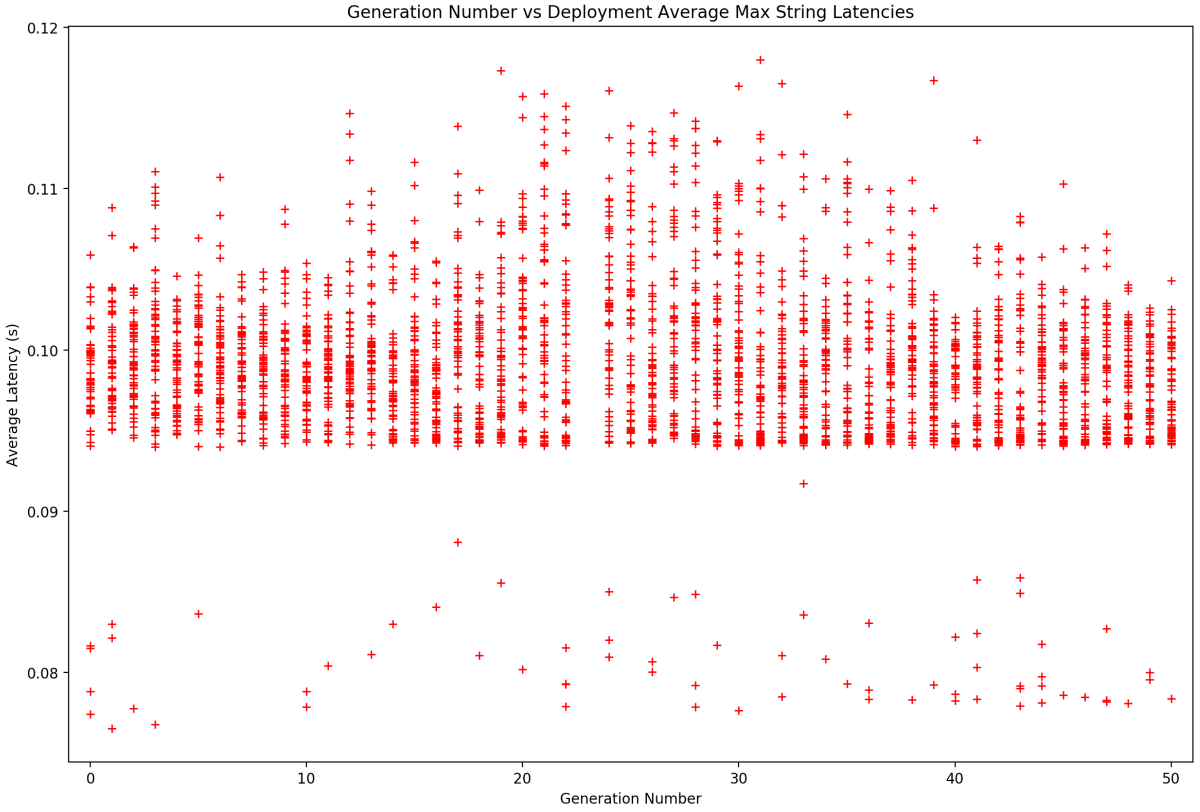

Figure 7.42: Design scenario 3: average measured string latencies for an undersea sensor software system (50 generations with populations of 60, and 58 samples for each population member)

These results show the influence of temporal performance requirements on the CPU resource utilisation across the nodes, compared with the resource level found with design scenario 2. In this case study, an increased number of nodes are executing beyond their required resource threshold levels. Two nodes have a reduction in utilisation levels compared with design scenario 2, while one remains (as expected) at the same level as design scenario 2.

It can be seen that Cranberry01 has had an increase in utilisation level to approximately 17% but remains under its threshold, Cranberry02 has increased its level to approximately 30% and has exceeded its threshold, while Cranberry03 has also increased its utilisation level to approximately 15% and also exceeded its threshold level. The results for Mandarin01 see a reduction in utilisation level to approximately 10% and it is below its threshold, while Mandarin02 also sees a reduction in utilisation level to approximately 65% but it is still exceeding its threshold level.

Lastly, it can be seen that the utilisation level for Mandarin03 is the same, which is to be expected as this is the node that has a single software component deployed to it throughout the case studies. While the utilisation graph seems to be more variable than would be expected with the constant workload deployed, the generation population members CPU utilisation graph (Figure 7.50), shows this is due to some outlier utilisation measurements. The vast majority of the measured results exhibit the expected behaviour and are more aligned with the deployed constant workload.

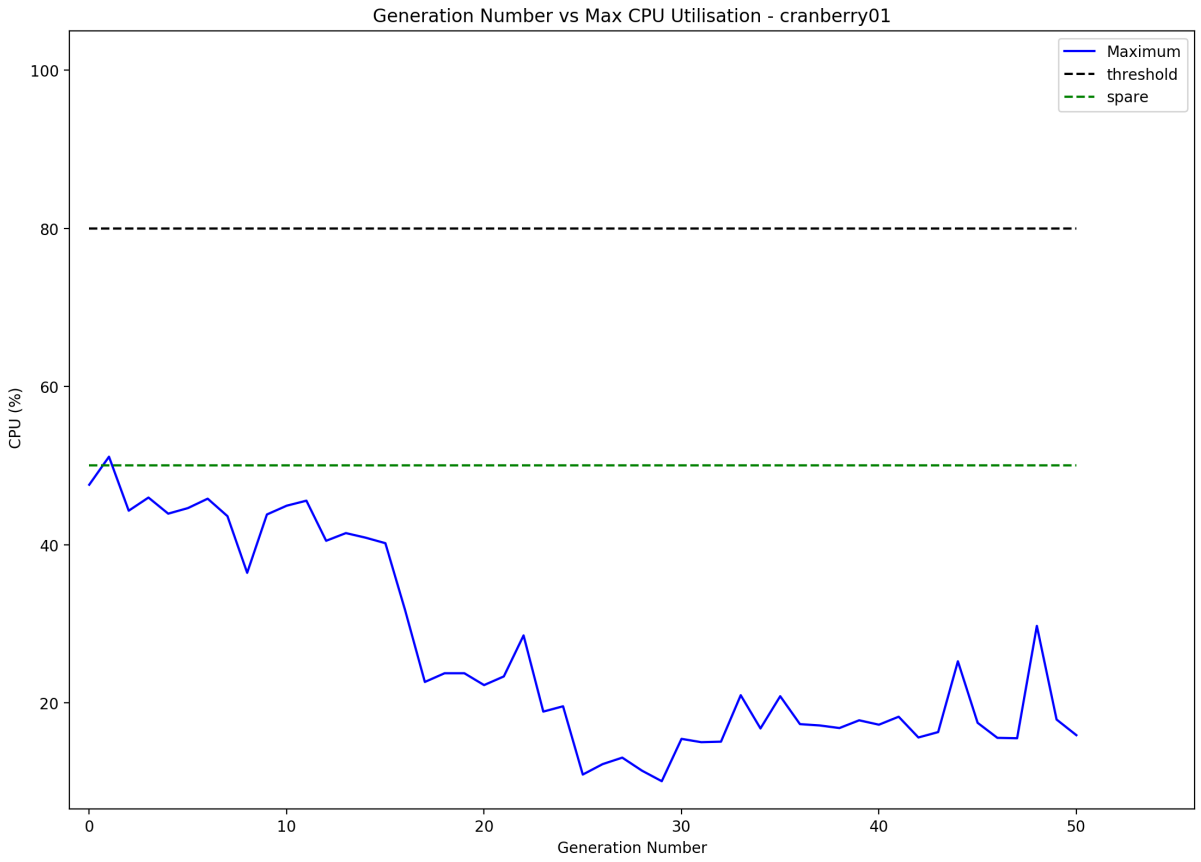

Figure 7.43: Cranberry01 largest maximum CPU resource consumption for an undersea sensor software system for the design scenario 3 experiment (50 generations with populations of 60, and 58 samples for each population member)

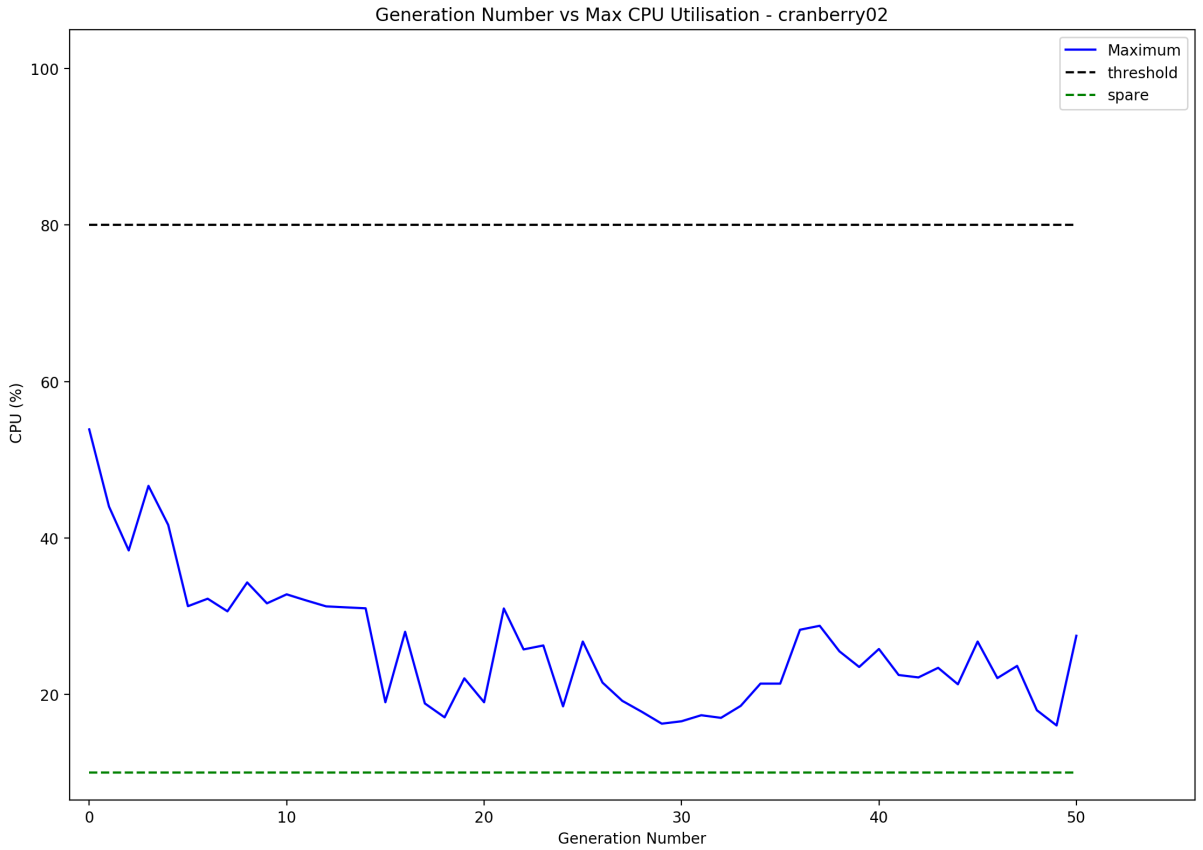

Figure 7.44: Cranberry02 CPU largest maximum resource consumption for an undersea sensor software system for the design scenario 3 experiment (50 generations with populations of 60, and 58 samples for each population member)

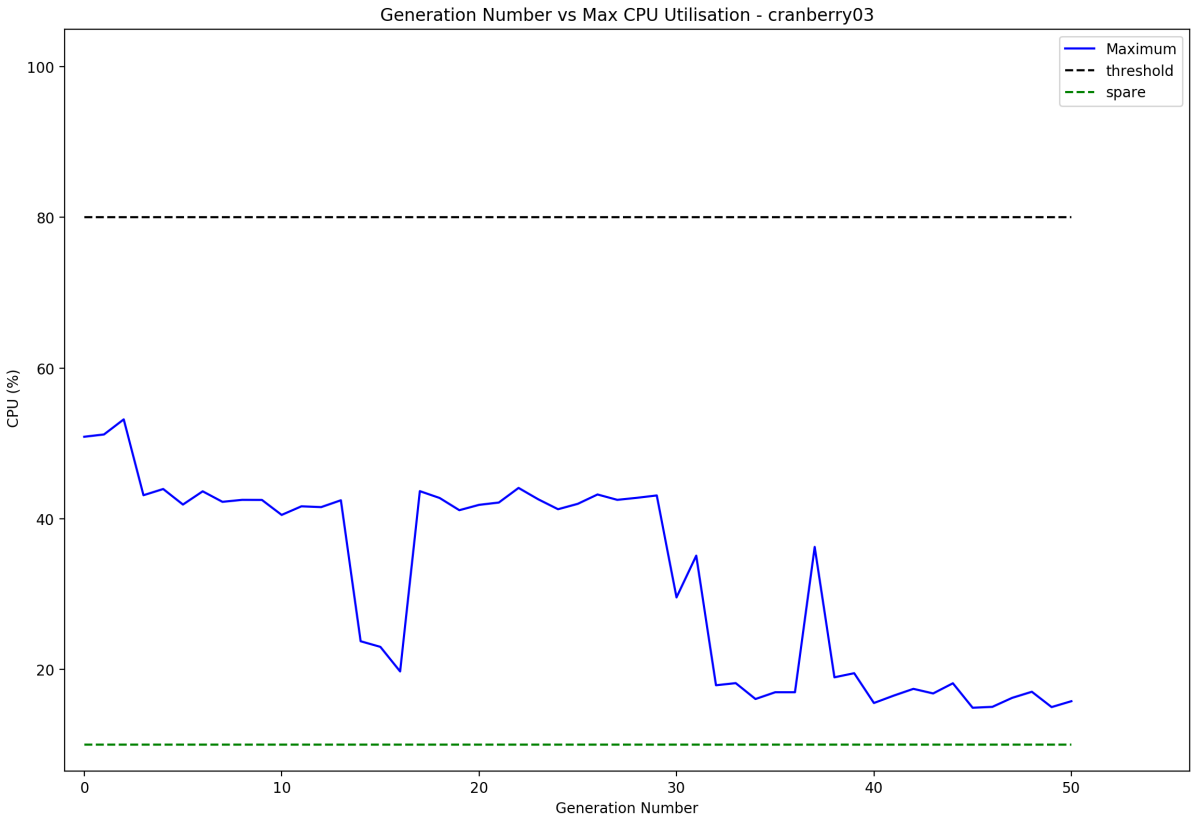

Figure 7.45: Cranberry03 CPU largest maximum resource consumption for an undersea sensor software system for the design scenario 3 experiment (50 generations with populations of 60, and 58 samples for each population member)

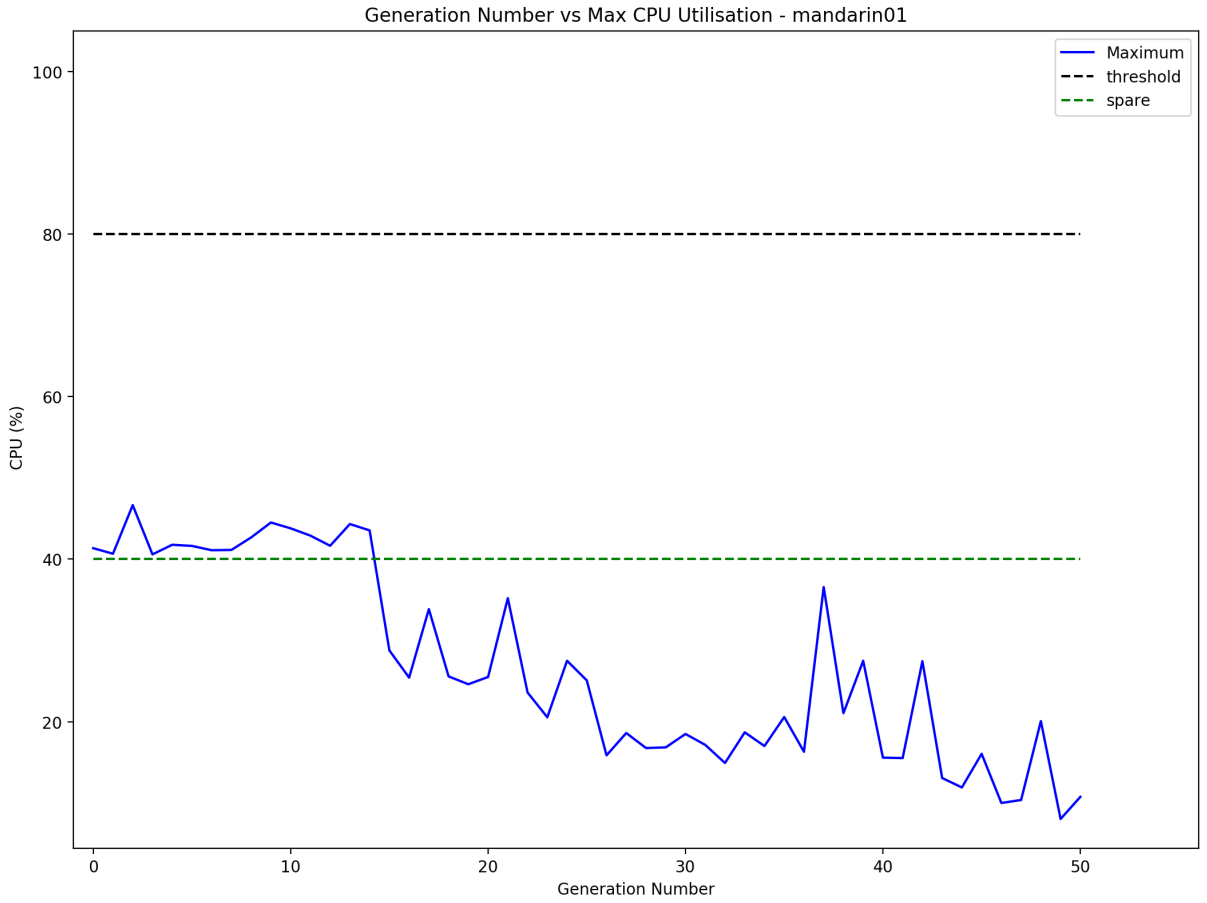

Figure 7.46: Mandarin01 largest maximum CPU resource consumption for an undersea sensor software system for the design scenario 3 experiment (50 generations with populations of 60, and 58 samples for each population member)

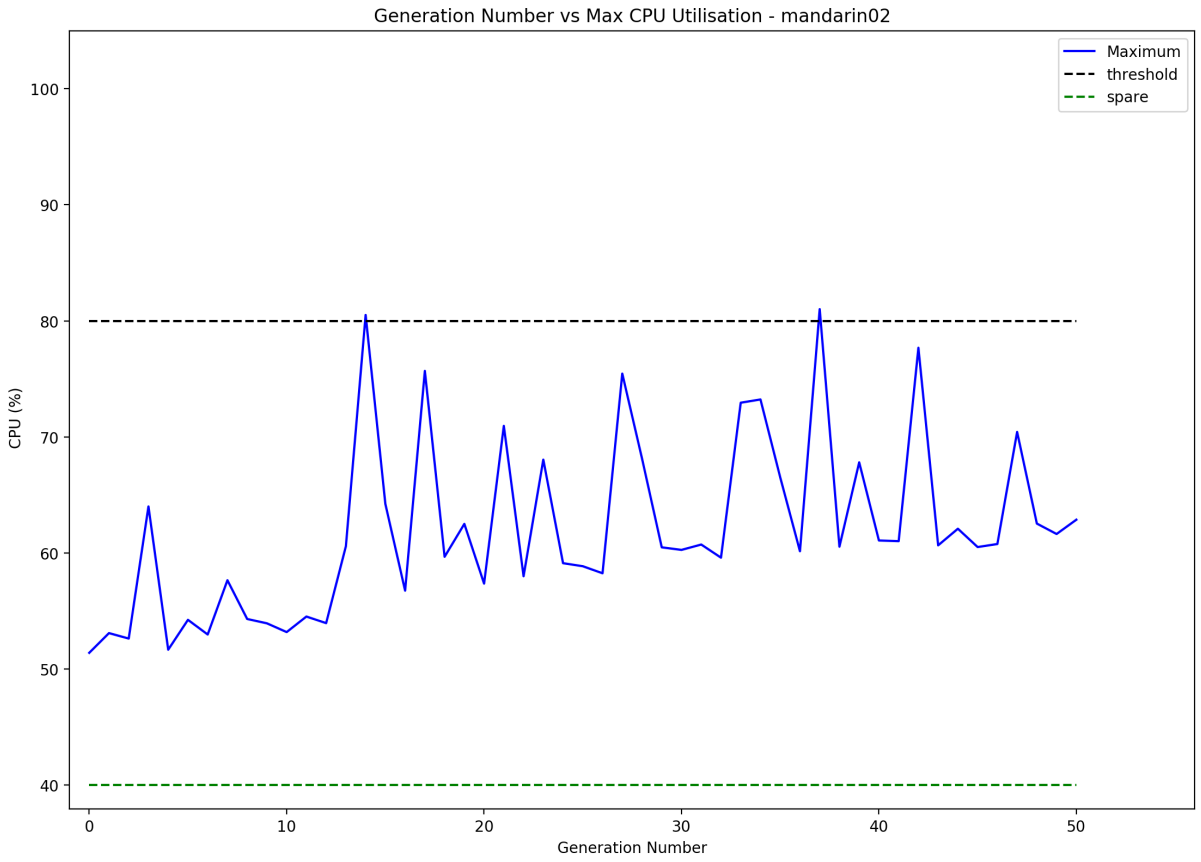

Figure 7.47: Mandarin02 largest maximum CPU resource consumption for an undersea sensor software system for the design scenario 3 experiment (50 generations with populations of 60, and 58 samples for each population member)

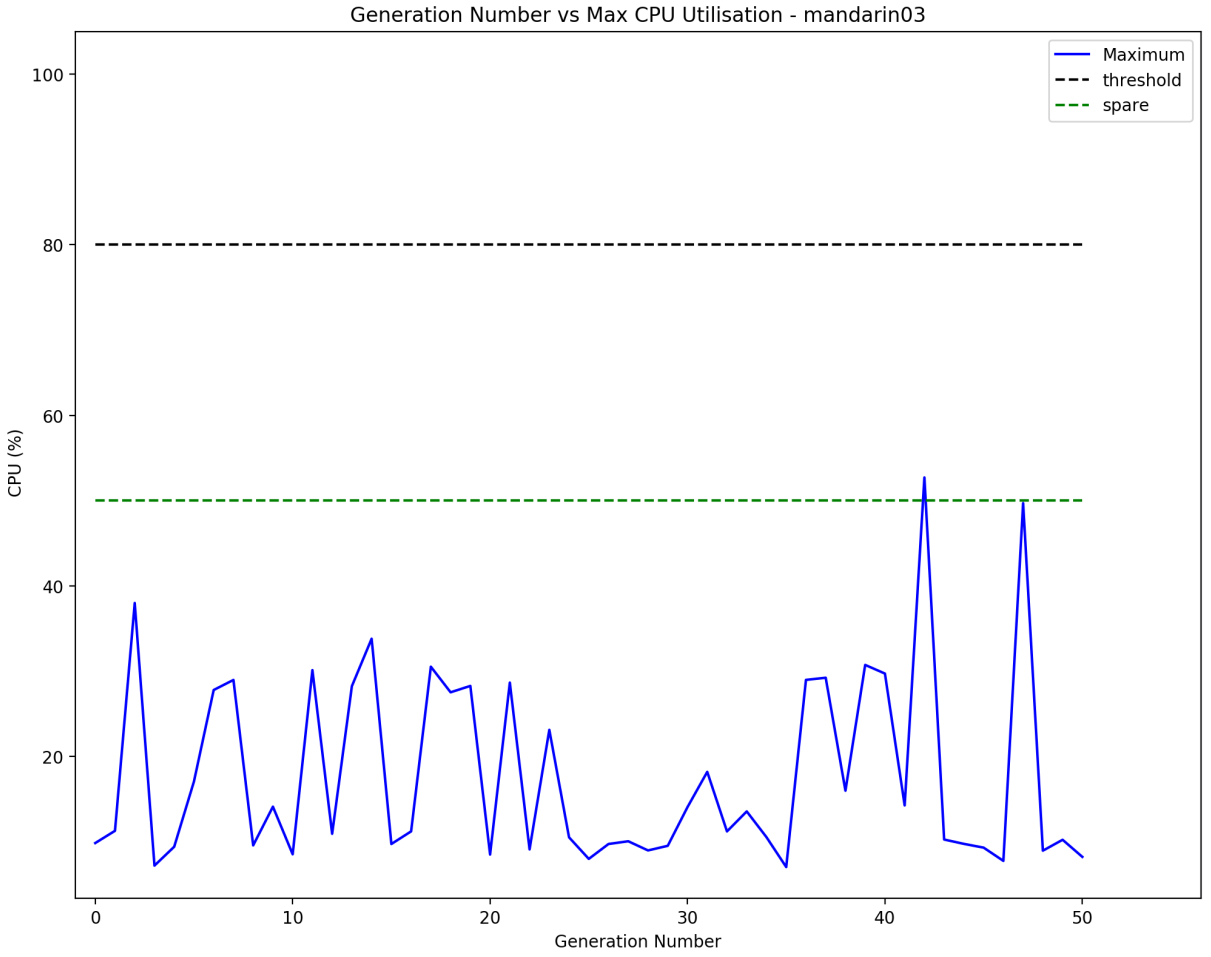

Figure 7.48: Mandarin03 largest maximum CPU resource consumption for an undersea sensor software system for the design scenario 3 experiment (50 generations with populations of 60, and 58 samples for each population member)

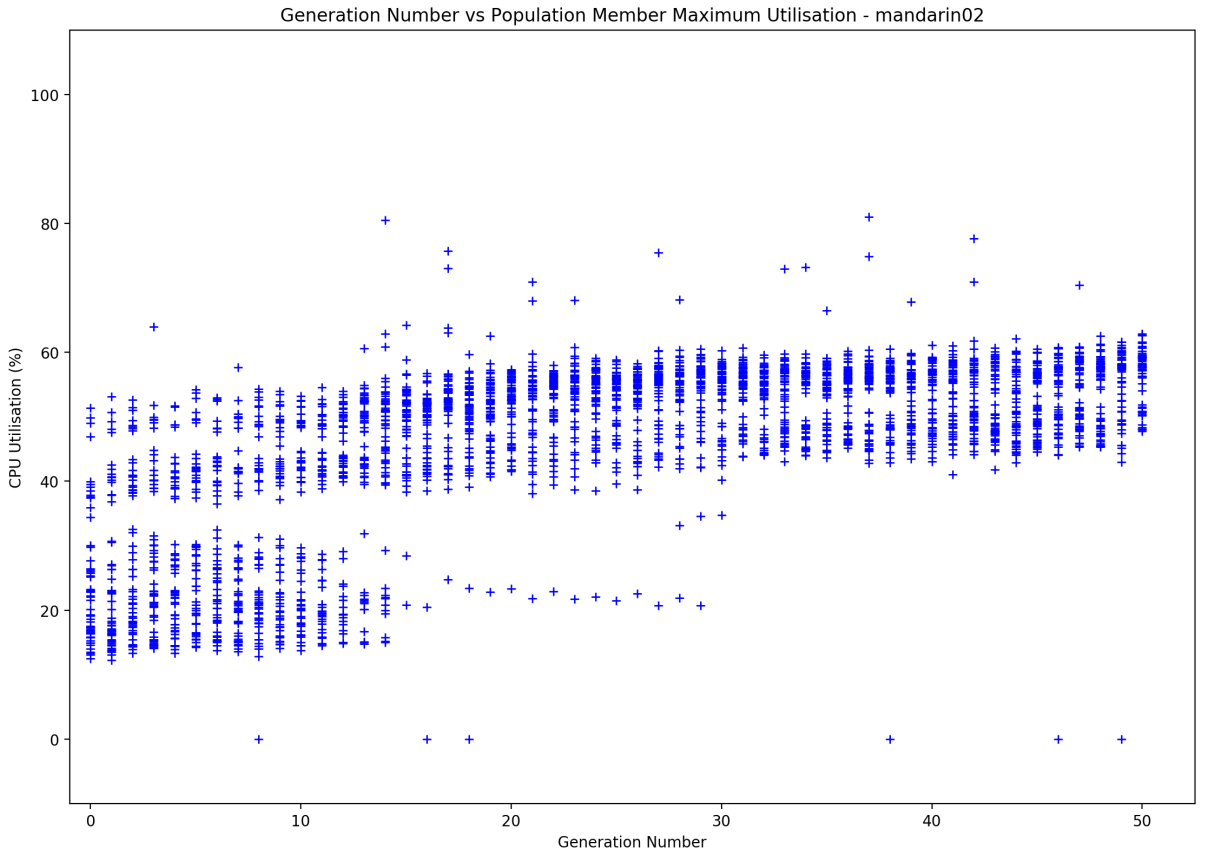

Figure 7.49: Mandarin02 maximum CPU resource consumption for population members for each generation for the design scenario 3 experiment (50 generations with populations of 60, and 58 samples for each population member)

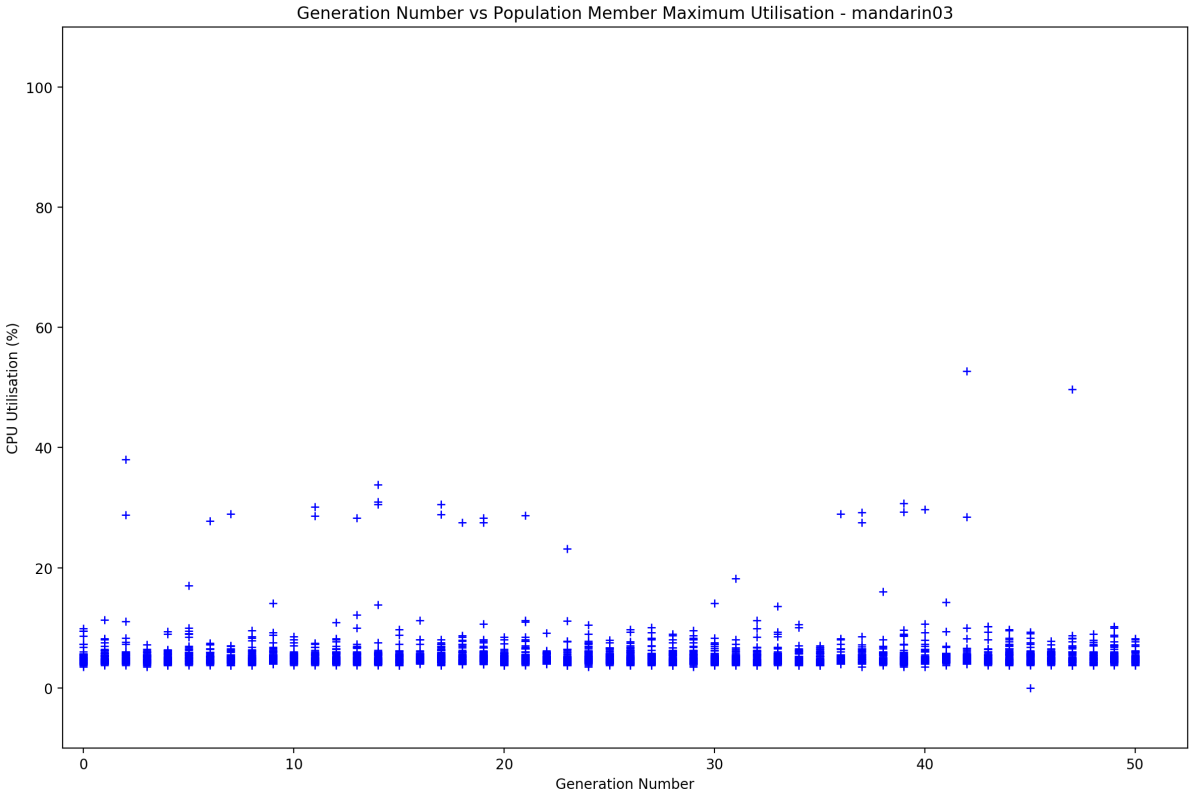

Figure 7.50: Mandarin03 CPU maximum resource consumption for population members for each generation for the design scenario 3 experiment (50 generations with populations of 60, and 58 samples for each population member)

7.4.2 Case Study Discussion

The baseline design scenario showed critical chain latency performance converging to a range of approximately 0.217 to 0.232 seconds, and a CPU utilisation distributed reasonably even across the six available computing nodes. Both items showed the expected behaviours and results. From the best performing objective score, the optimisation process identified nine unique deployment options capable of providing the best performing latency performance for the identified critical chain. Reviewing the nine different deployments, the optimisation process chose computing nodes 2 and 3 (Seawater Sensor 3 and Seabed Sensor 4) as the main deployment points for the various levels of workload to deploy. Furthermore, the objective score is a direct indication of the critical chain latency performance, and was 349.53.

Design scenario one introduced the design changes requiring the system design to ensure mandated deployment constraints are satisfied between the identified software components and computing nodes, alongside the no co-location constraint between the two identified software components. As a result of satisfying these constraints it was observed that the convergence performance was reduced and 37 unique deployment profiles were identified, compared with the nine from the baseline scenario. As with the baseline scenario, the objective score was directly related to the critical chain latency, was slightly increased (358.60) and produced a slightly reduced latency range of approximately 0.215 to 0.225, because of the increased number of deployment options identified for the best performing objective score.

Reviewing the identified unique deployment profiles, we observed the majority of computing nodes chosen for software component deployment shifts from 2 and 3 (Seawater Sensor 3 and Seabed Sensor 4) for the baseline scenario to 1 and 2 (Seawater Sensor 2 and Seawater Sensor 3) for scenario 2. It would be reasonable to conclude this shift is based on the highest workload being deployed to computing node 3 and the aim being to minimise the impact of temporal delay from high CPU utilisation for this node. This would be further driven by the overall aim being to minimise the CPU utilisation across the available computing nodes, as well as satisfying the deployment constraints (which includes only having a single software component on the computing node 5).

Design scenario two introduced the CPU spare resource constraints (in response to power consumption reductions). Addressing these constraints, the optimisation process identified a single deployment profile related to the best performing objective score. Furthermore, the objective score was significantly reduced compared with the baseline scenario (and second design scenario) as a result of the optimisation algorithm producing highly optimised results (better than required), leading to an overall reduction. However, we observed limitations in the optimisation process, where highly optimised solution were found at the expense of a non-satisfactory solution to one of the defined constraints.

Reviewing the single deployment option identified, we observed the same component deployment to computing node 3 as was found with the previous scenario. We also observed that computing node 4 is the primary node for software deployment, which was reflected in the CPU utilisation graph and a reflection of

the how the reduced objective score was achieved. We also observed the critical chain latency range increased from approximately 0.215 to 0.255 seconds (compare to the baseline), which indicates the optimisation algorithm was less focused on minimising the critical chain latency as a consequence of trying to improve CPU utilisation performance.

The final design scenario introduced latency performance requirements for a pair of interfaces and a string of interfaces (a segment), both within the critical chain of interest. In addition to the previously introduced requirements and constraints, we observed the optimisation process identified a single deployment profile, but the companion objective score was significantly higher than for the baseline scenario and subsequent scenario one and two. While this scenario satisfied the defined constraints, the increased objective score reflect the fact that CPU utilisation and the temporal performance threshold were not satisfied.

Reviewing the deployment profile, we also observed the change in deployment profile only considered the software component associated with the defined interface and string temporal performance requirements. The deployments found from the previous scenario were maintained, along with the inherited performances. We also observed the critical chain latency range was similar to the baseline scenario, which indicates that the optimisation algorithm is focused on temporal performance, this time at the expense of CPU utilisation, which was reflected in the observed over-utilisation of some of the available nodes.

Overall, we saw the successful application of the optimisation capability in gaining insights into design options for an undersea sensor system and its de-

ployment within the undersea environment to achieve defined performances. We observed the optimisation process works through the increasingly complex solution search space with the layering of competing requirements and constraints, and give key insights on performance outcomes through the design scenarios. The approach provided details of the difficulties in meeting defined thresholds, where flexibility lies for adjustment with over-exceeding performances being obtained, and, in the case of the last design scenario, the insight that not all performances will be met.

The result of this is that the system designer is equipped to build on the insights gained by refining design decisions and adjusting performance expectations. This will then flow back into the optimisation process until the undersea system design and identified deployment delivers performance for the current version of the system to a level acceptable to the system designer.

An extension to gaining insights into the current version of the system design, deployment and performances, would be to use a language and modelling approach to enable the optimisation process to consider software system evolution. This would enable an understanding of how current requirements and constraints would be impacted by software system evolution, and provide insight into what adjustments would be required (and their resulting performance) in response to any such software system evolution.

In the case of the undersea sensor system, the ability for the optimisation process to consider software footprint growth, degradation of batteries through age, or architectural changes would provide great benefit. It would enable system designers to understand how far the undersea sensor system's current computing

infrastructure could support any future sensor system software development, and what requirements and constraints may need to be changed to support any such evolution.

8. Evolution DSML

As indicated within the Aims chapter, the focus of the DSML and associating modelling framework is to allow system designers to model and explore how certain elements of a current system design may evolve. The benefit of such an approach was also discussed in the Case Study, where insights could be gained into how best to evolve and what optimised performances would be achieved for the undersea sensor system.

By using a newly-created DSML, model information can be held on the behaviour and characteristics of the ways in which the system may evolve. Combining this with information from the extant system design model, integrated with a SEM environment, system designers can explore optimal design changes and performance impacts for evolving systems.

8.1 Component Evolution Modelling Language (CEML) Model

The CEML (Component Evolution Modelling Language) DSML was developed using a Meta-DSML and GME (Davis, 2003) development approach. The modelling needs for this DSML were as follows:

- Define software footprint evolution paths
- Define architectural evolution paths
- Define entity relationship constraints
- Define computing resource evolution paths
- Define temporal performance constraints

This approach consisted of using a meta-modelling language based on the UML class diagram notation and OCL constraints to specify the new modelling paradigm of the application domain. Using the meta-modelling language to define syntactic, semantic, and presentation information, a DSML meta-model is created. This DSML meta-model is then applied to the GME environment to automatically generate the target domain-specific modelling environment, which then allows for the construction of the domain-specific models. These models then enable definitions associated with the application domain, such as what relationships may exist among those application concepts, how the concepts may be organised and viewed by the modeller, and the rules governing the construction of models.

Figure 8.1 shows one of the UML class diagrams produced to create the CEML DSML. In this case, it defines the relationships between modelling elements at

the component level, alongside evolution characteristics and behaviour definition. Within this UML model, there are connection points to other UML class diagrams within the CEML DSML via *ModelProxy* elements. These are the *ComponentArchitectureInstanceProxy*, *DecomposeProxy* and *ComponentInstanceProxy* *ModelProxy* elements. Numerous *Atom* elements are found, indicating these model elements are at the lowest level of modelling and definition, while the *Model* elements indicate a lower level of modelling is present within this area, and the composition of the next level is based on those elements connected to the *Model* UML element. There are two *Connection* elements to define the rules as to which model elements can be connected, while a *FCO* (or First Class Object) UML element provides abstraction and enables inheritance across the different objects.

Appendix F shows all of the UML class diagrams found as part of the CEML DSML.

The CEML meta-model was developed to allow for modelling context views for evolution characteristics and behaviour modelling at the architectural, component and resource layers. It also provides a context view to allow for deployment constraint definitions. The complexities associated with modelling via numerous context views were also reduced by separating each into GME's 'aspect' model view, thereby removing the need to capture all information within a single modelling canvas. The three main modelling layers for CEML are the *ComponentArchitectureInstance*, *ConstraintDeployment* and *SubComponent_Definition*. Within these layers exists a combination of different level modelling canvases and modelling aspect windows.

The entry point to the CEML modelling stage was the importation of a minimum set of details from the PICML system design model to enable representation of the system design within the CEML modelling environment. Figure 8.2 shows the CEML modelling environment, with an example of an imported, simple, three component software system for an unmanned autonomous vehicle (UAV) system. The components comprised of a global positioning system (GPS) processing component (Position Marker 1), a vehicle control component (Position Marker 2) and a communication component (Position Marker 3). We can also see that this modelling layer consists of two different modelling aspect tabs for modelling. These are the *ComponentArchitecture* and *ComponentGrowthDefinition* aspects (Position Marker 4).

Figure 8.3 shows the CEML modelling environment in the *ComponentGrowthDefinition* modelling aspect view. Within this view tab, four different model entities (Position Markers 1-4) are available for modelling growth at the component architecture level. These entities allow the modeller to detail system design evolution in the areas of:

- Redundancy
- Workload Growth
- Decomposition

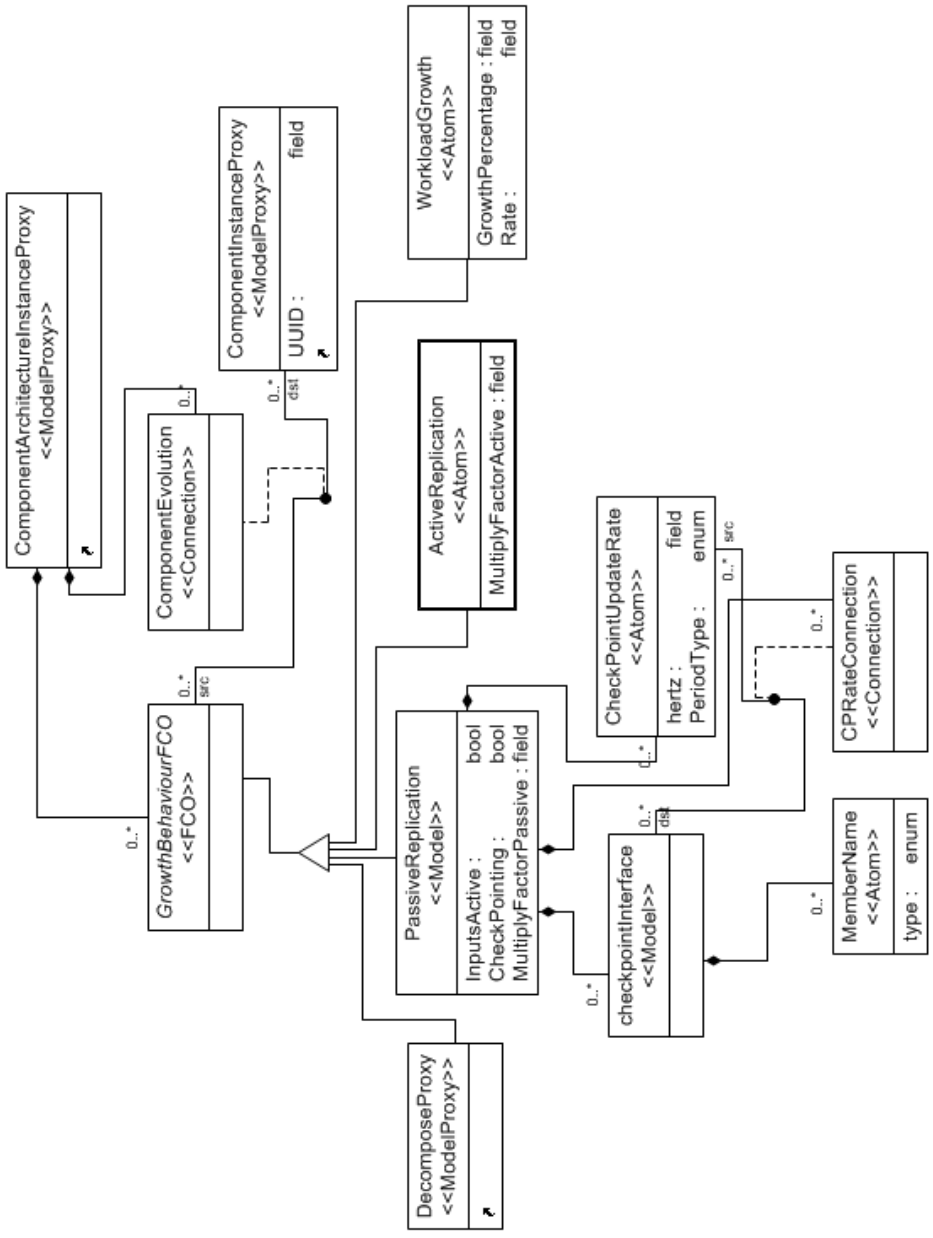

Figure 8.1: CEML class diagram

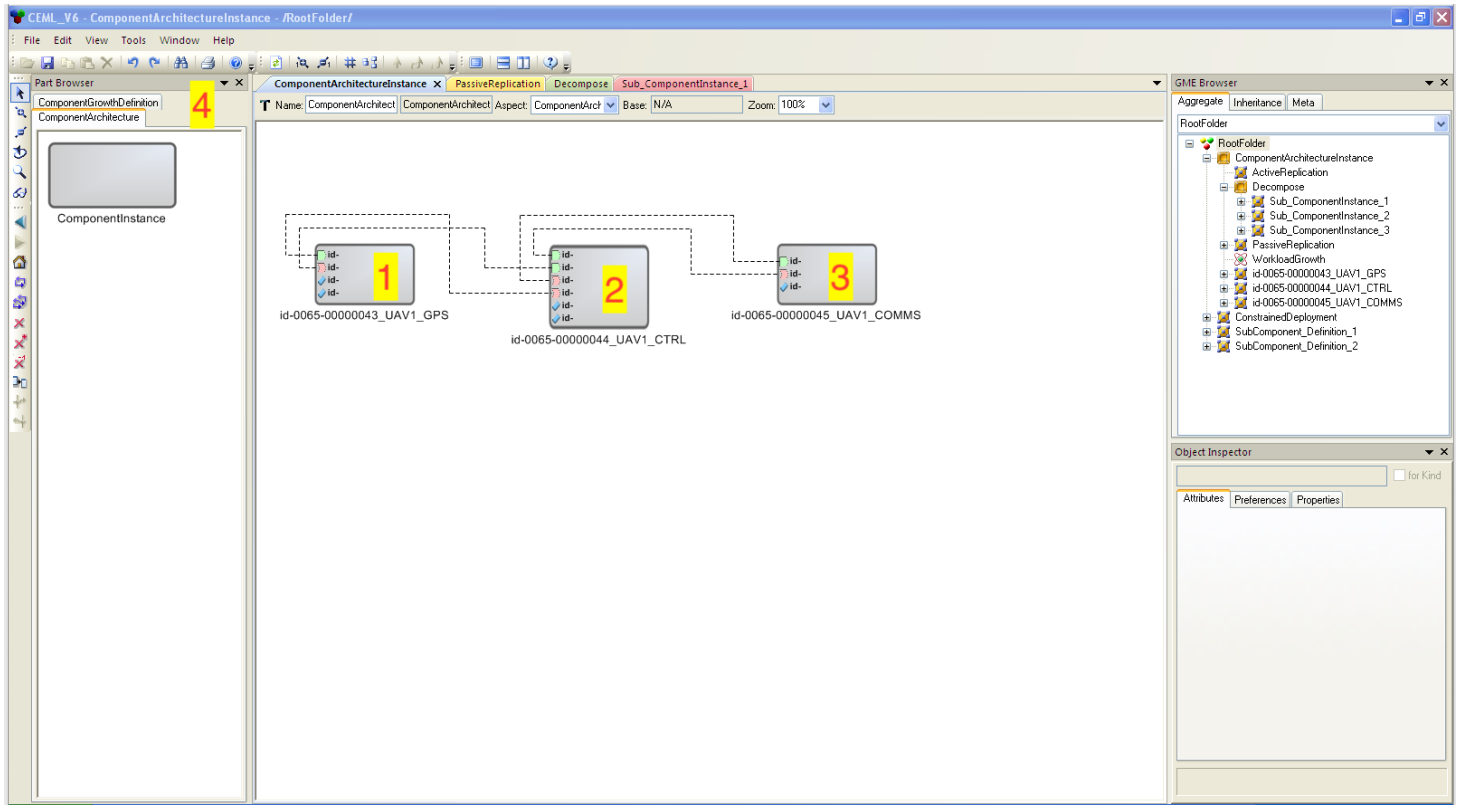

Figure 8.2: CEML modelling environment showing the imported system design

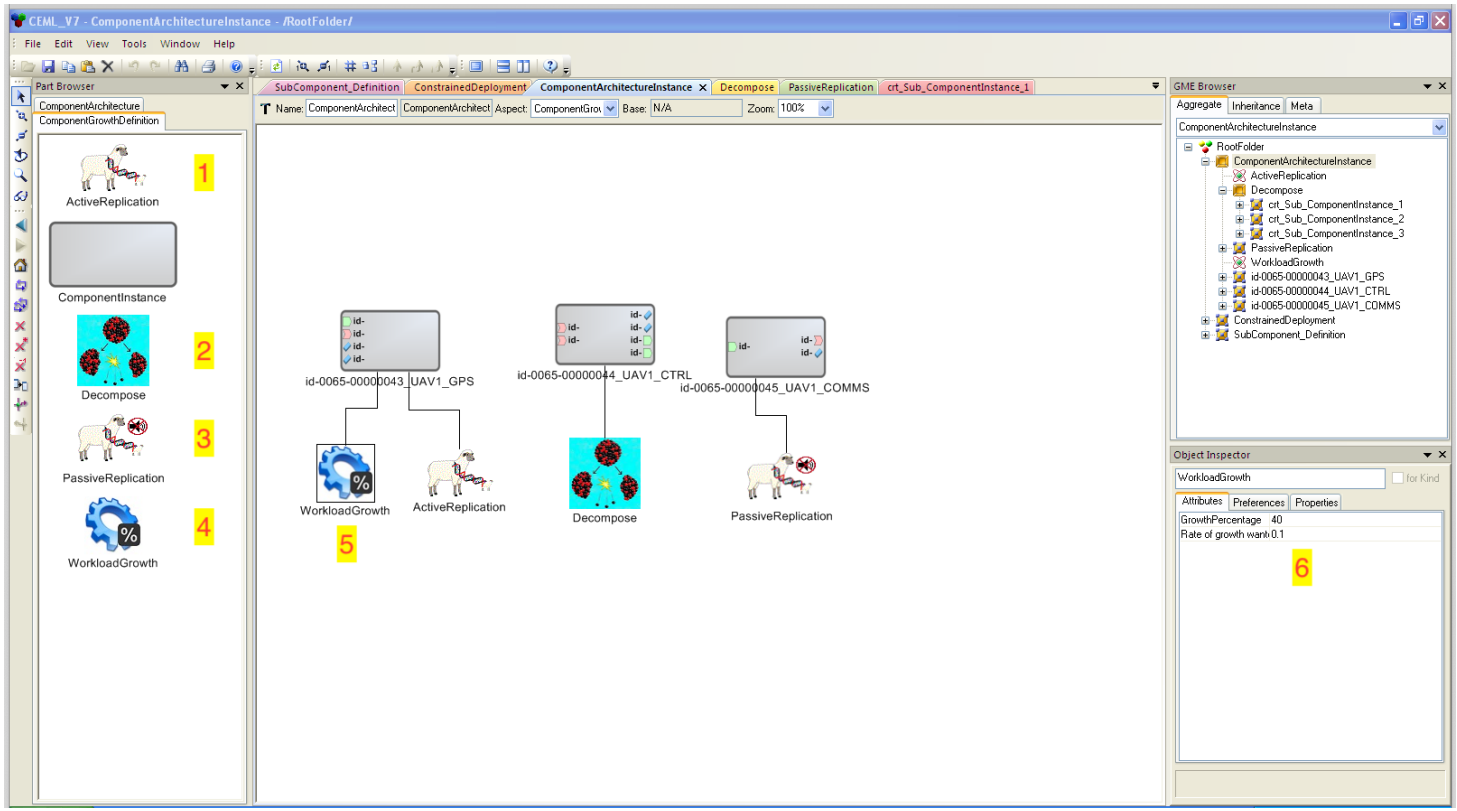

Figure 8.3: CEML modelling environment showing the component growth modelling

Redundancy

In the case of redundancy, the modeller is offered four different forms of replication as defined by NSWCDD, 2004. These are Primary/Shadow, Active, Passive and Check Pointing. To achieve any of these, the modeller is provided with both active (Position Marker 1) and passive (Position Marker 2) replication model entities, along with associated configuration items such as replication number settings.

Furthermore, passive replication modelling includes an additional modelling level to detail the characteristics of check pointing replications (if enabled). As can be seen in Figure 8.4, there are two model entities available to set the number of checkpoint interfaces (Position marker 2) and the check point message transmission characteristics (Position Marker 1) behind each interface. Once an interface is connected to an *CheckPointUpdateRate* model entity, the modeller can then choose the entity and define the frequency and whether increasing frequency is required (Position Marker 3 and 4).

Workload Growth

Figure 8.3 also shows the option to define a workload growth (Position Marker 4). Using this model entity, the modeller can assign a growth for workloads found within identified components (Position Marker 5). Once assigned, the modeller can then define the size of the growth and the number of steps[1] to reach that growth

[1] Each step equates to a new architecture configuration

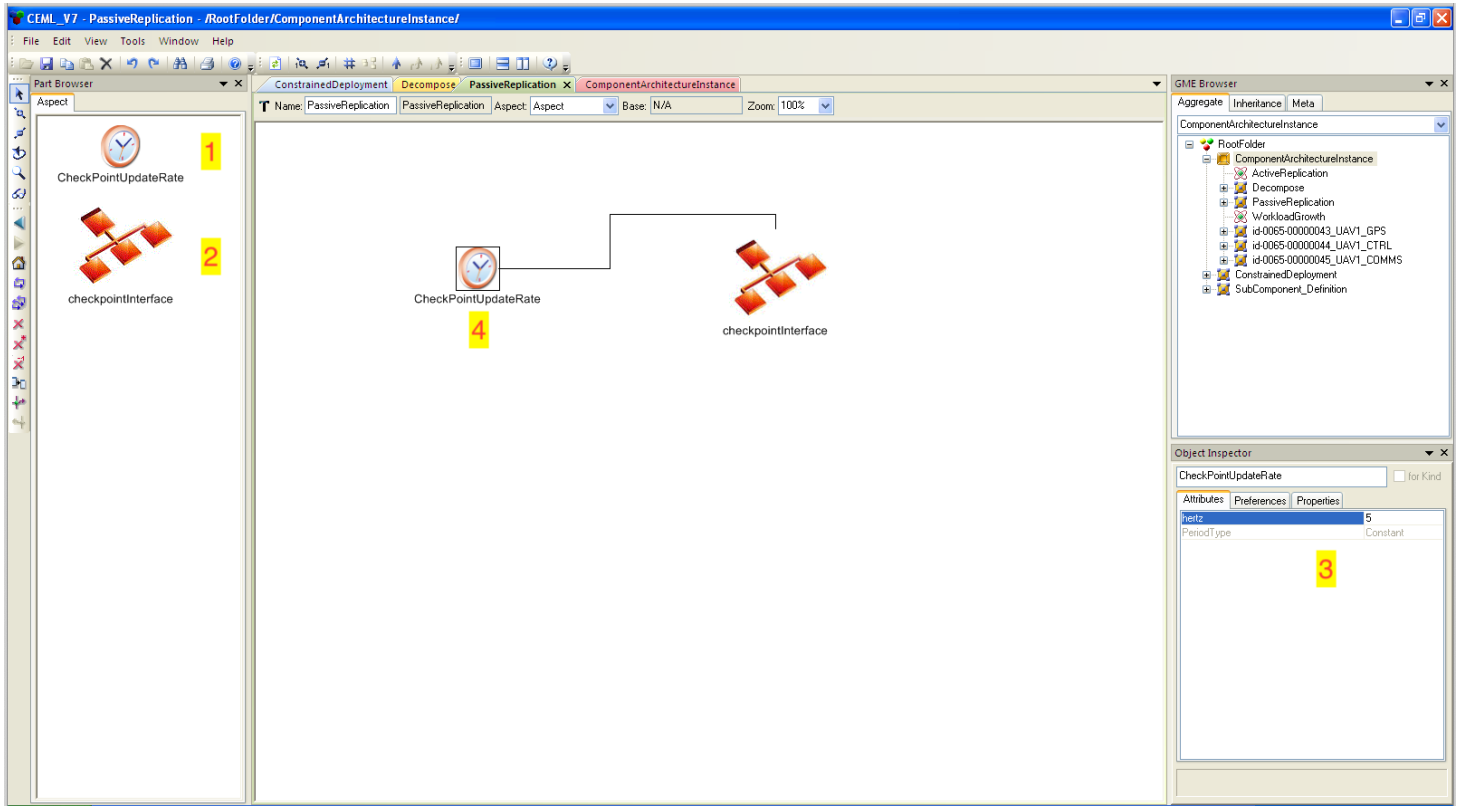

Figure 8.4: CEML modelling environment showing the passive replication checkpointing modelling

(Position Marker 6) for the workloads within the component. The last model entity is the *Decompose* model entity (Position Marker 2). This allows the modeller to investigate particular components moving from a monolith to being decomposed into a number of sub-components. In this case, the 'id-0065-00000044_UAV1_CTRL' component from the imported PICML UAV system design model has been selected.

Decomposition

Figure 8.5 shows the component decomposition architecture modelling view and details the decomposition architecture of the *id-0065-00000044_UAV1_CTRL* chosen for decomposing in Figure 8.3. In this modelling view, the number of sub-components wanted for the decomposition is defined along with how the connections between the sub-component interfaces are required. In this case, the decomposition consists of three sub-components and three new connections between five new sub-component interfaces (Position Markers 2 and 3). While the connections between new sub-component interfaces occur in this modelling view, the interfaces and event behaviour within each created sub-component are required and defined via the sub-component interface and behaviour modelling view.

Figure 8.6 shows the interface and event behaviour modelling for the sub-component *crt_Sub_ComponentInstance_3* from Figure 8.5. In this modelling view, all the model entity options available for interface and event behaviour definitions can be seen in the Part Browser (Position Marker 1).

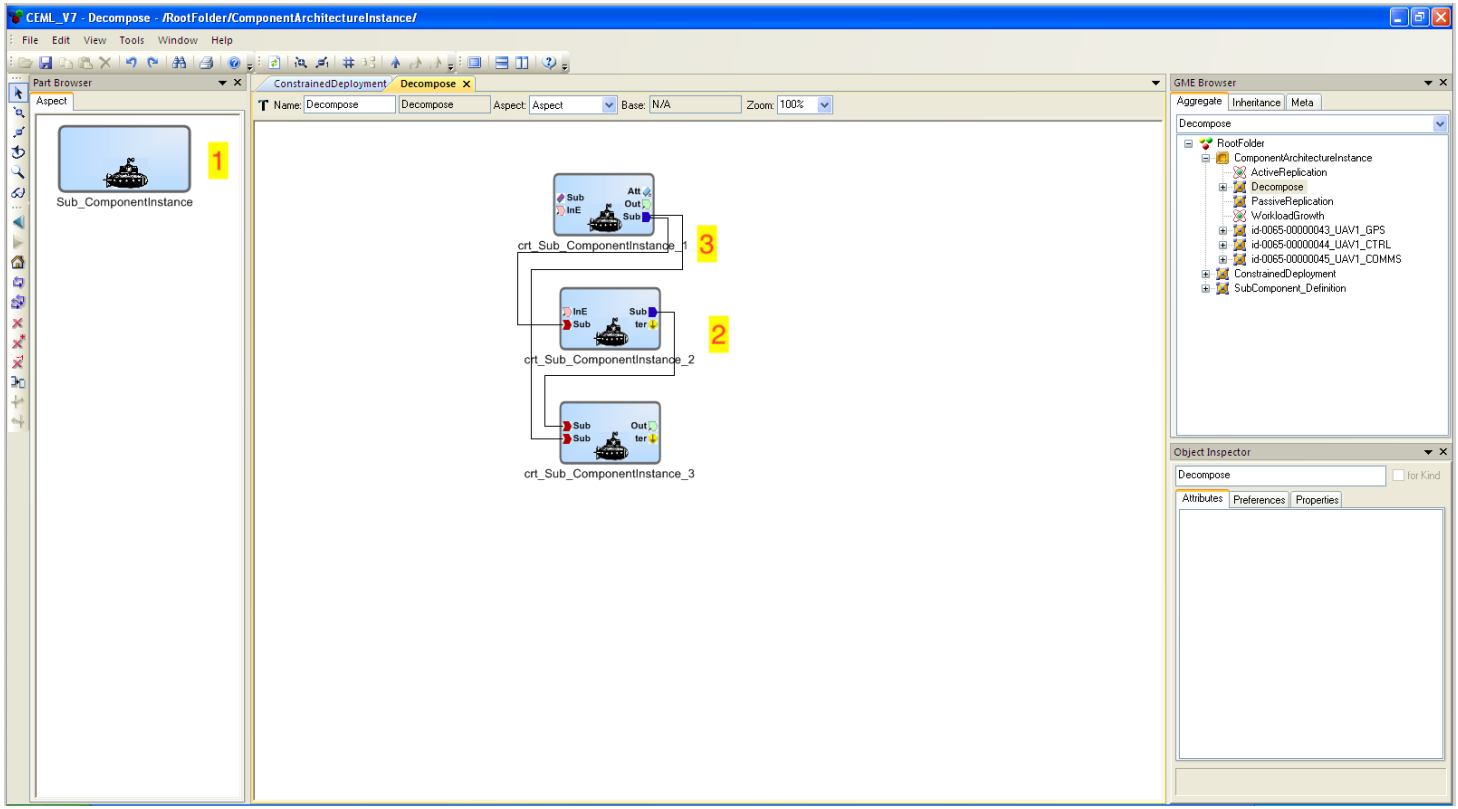

Figure 8.5: CEML modelling environment showing the component decomposition architecture modelling

The main modelling window shows the definition for two event threads, both beginning from newly-created sub-component input event port interfaces and ending with a connection to an original *id-0065-00000044_UAV1_CTRL* output event port interface (Position Marker 3), while the other is terminated internally. A branch condition (Position Marker 2) can also been seen, leading to the creation of two different workloads being executed within this sub-component. The last part of this modelling phase is to highlight a 'Sub_Workload' model (Position Marker 4) and then define the characteristics of that workload (Position Marker 5). The settings for workloads within sub-components allows for definitions of the size of the workload relative to the original monolith component workload amount, whether this workload level is to be reach through a series of steps[2] and if logging is required for the workload event.

Once the component architecture growth and required component decomposition modelling is complete, the modelling moves to the deployment constraint modelling. To reduce complexity, aspect windows are used to guide the modeller through the numerous constraints that can be modelled for the evolved system design deployment. These modelling aspects allow for:

- Computing Node Mandate Deployment
- Component No-co-location
- Minimum Temporal Performance
- Resource Capacity

[2]Each step equates to a new architecture configuration

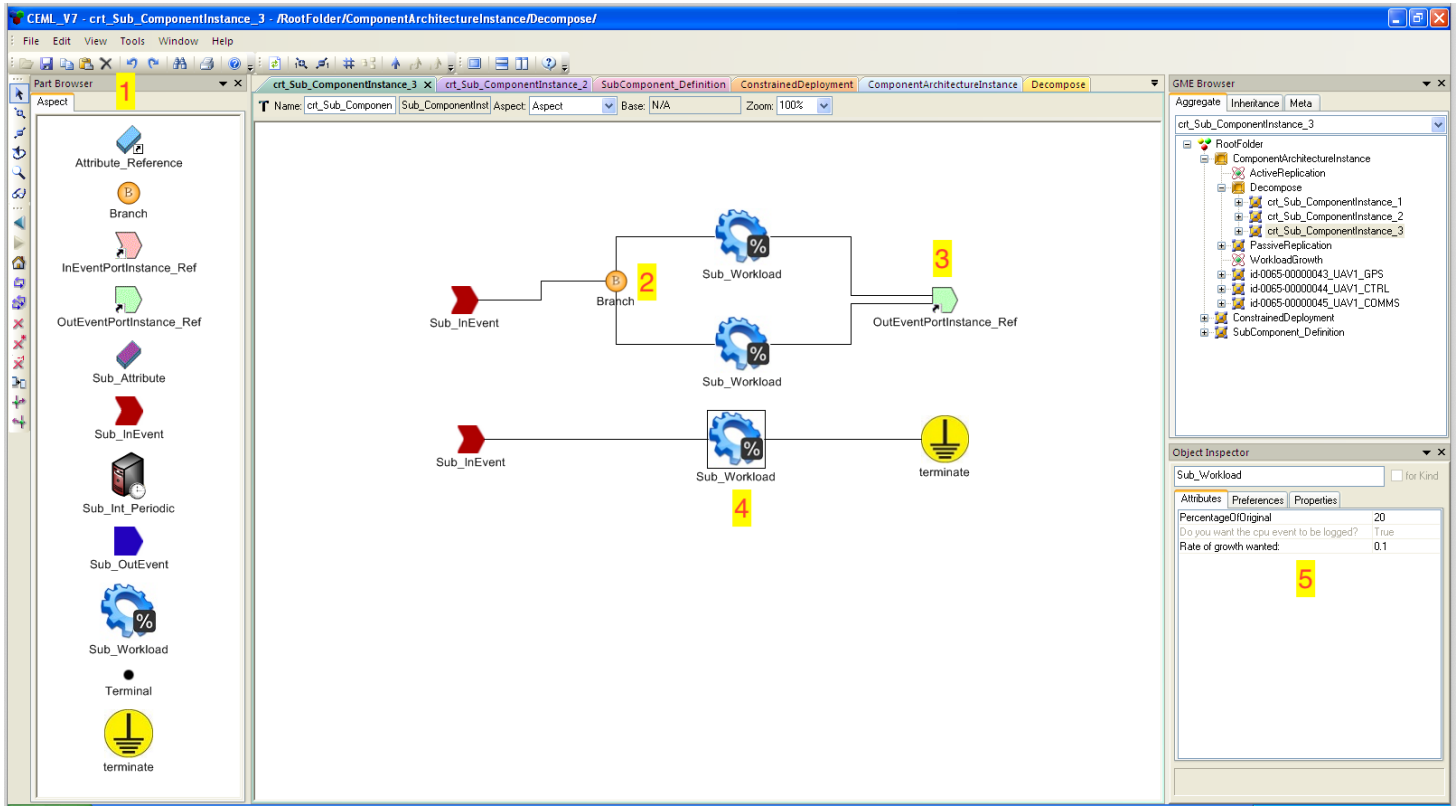

Figure 8.6: CEML modelling environment showing a sub-component interface and event behaviour modelling

Computing Node Mandate Deployment

Figure 8.7 shows the mandate deployment constraint modelling environment. This modelling view has four modelling entities for use: two to account for the system components (Position Marker 1) and sub-components (Position Marker 4), and two to account for the computing nodes available for deployment. In the case of the two computing nodes related to modelling entities, one identifies a particular computing node (Position Marker 3) available for deployment, while the other modelling entity is the deployment group (Position Marker 2) that captures particular components and components requiring mandate deployment, and can be attached to one or more available computing node. In the case of more than one computing node being attached to a deployment group, all permutations of deployments are explored. In this modelling view, all the modelling elements within the main modelling window are greyed out and the modelling approach sees highlighting of all the parts of the system design associated with the mandated deployment.

In the scenario shown in Figure 8.7, one of the sub-components (Position Marker 7) of the *id-0065-00000044_UAV1_CTRL* is associated with a deployment group (Position Marker 6), which is also associated with a single computing node (Position Marker 5).

Component No-co-location

Related to modelling mandate deployment constraints, a modelling option is also available for no-collocation constraints between certain elements of the evolved

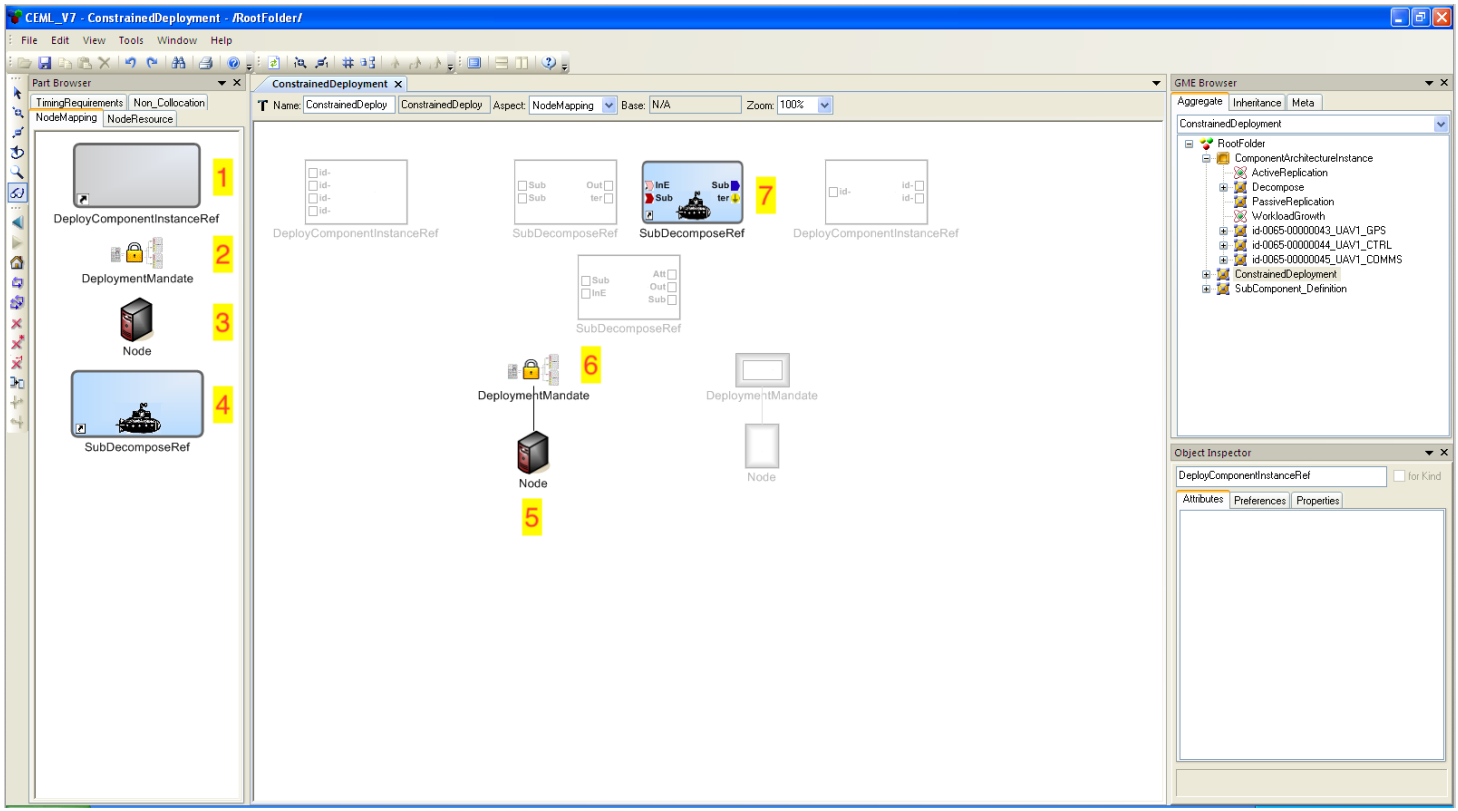

Figure 8.7: CEML modelling environment showing mandated deployment constraint modelling

system design. For these modelled constraints, the relationship is between the software components of the design and dictates that particular software components never reside together on an available computing node. As depicted in Figure 8.8, there are four modelling entities to identify the particular part of the evolved design (Position Marker 1, 2, 4 and 5) the constraints are to be attached to, while one modelling entity is used to define the constraint (Position Marker 3).

The main modelling window in Figure 8.8 shows two no-collocation constraints modelled for the UAV software system. One constraint (Position Marker 6) declares the two new sub-components cannot reside together when deployment occurs. The other constraint definition (Position Marker 7) defines that, as part of the construction of the passive replication for the UAV *COMMS* component, none of the redundancy components constructed to achieve the desired replication can reside with each other.

Minimum Temporal Performance

Following deployment modelling, the CEML modelling environment then defines temporal performance constraints for certain parts of the evolved system design. In this case, the modelling is defining maximum latency for connections between the software components of the evolved software system design. Figure 8.9 shows the connection between the original *id-0065-00000043_UAV1_GPS* component and one of the new sub-components that requires a temporal performance constraint (Position Marker 4). This also includes the definition of the maximum latency in milliseconds (Position Marker 5).

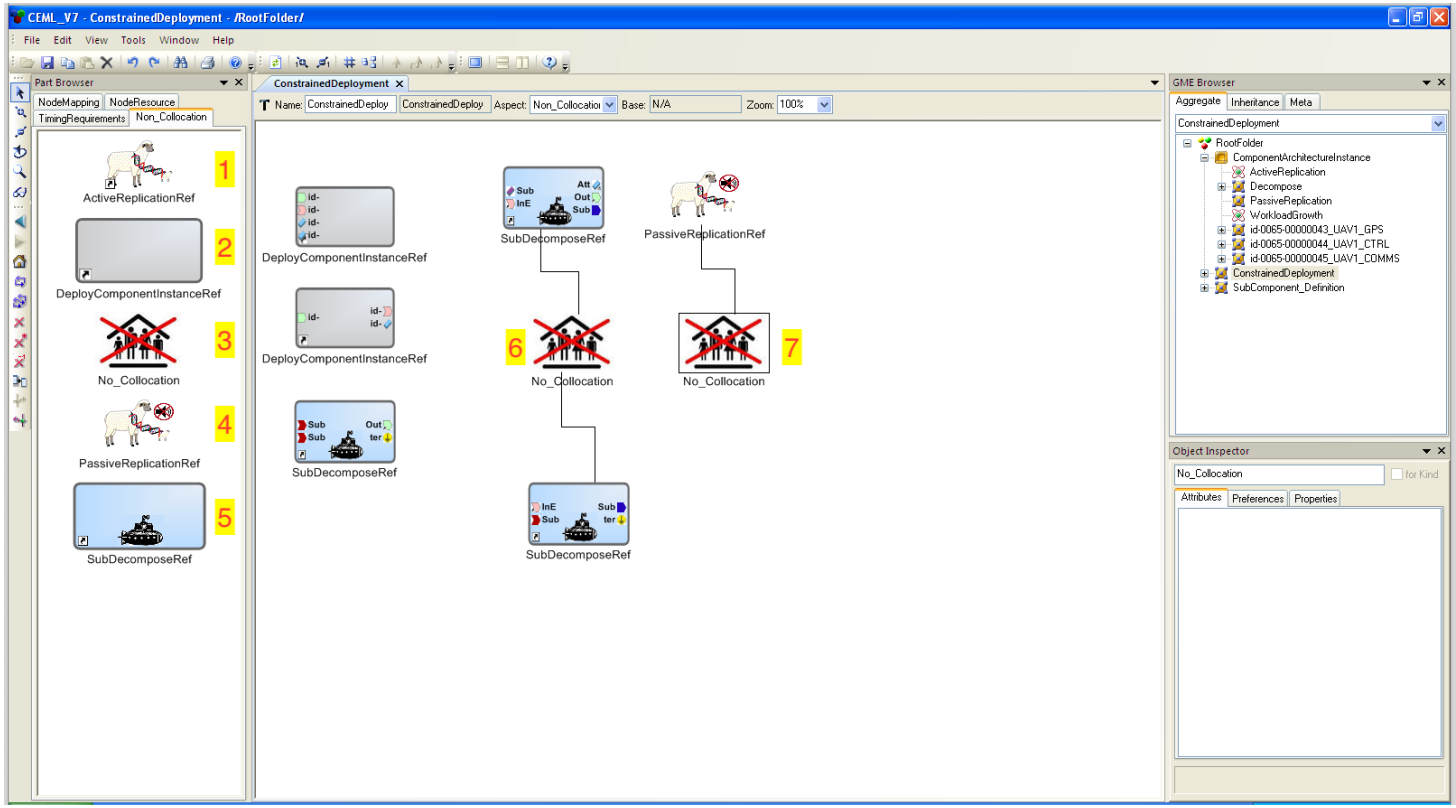

Figure 8.8: CEML modelling environment showing no-collocation constraint modelling

For this part of the CEML modelling process only components or sub-components are considered for temporal performance declaration. Future research would also look to add the replication elements constructed through the CEML modelling process.

Resource Capacity

The last area of constraint modelling within the CEML modelling environment defines the resource capacities constraint for CPU or memory resources that will be applied to the available computing nodes. Figure 8.10 depicts the modelling environment for defining the spare resource capacity for the CPU (Position Marker 1) or the memory (Position Marker 2) applied to the computing nodes. In this case, it can be seen that both computing nodes available within the UAV system design have spare resource constraints declared for both CPU and memory, and the values of these constraints are defined in the attribute declaration area (Position Marker 5).

Using all the architecture evolution characteristics and behaviours defined through the CEML modelling phase, the CEML Model Interpretation process (Figure 8.11) works through all the architecture evolution declarations and identifies the resulting evolved system designs. For this concept development, these new evolved system designs are based on permutations of all the architecture evolution definitions. Future research would consider the use of evolutionary algorithms to search for the most likely architecture evolution paths and resultant evolved system designs.

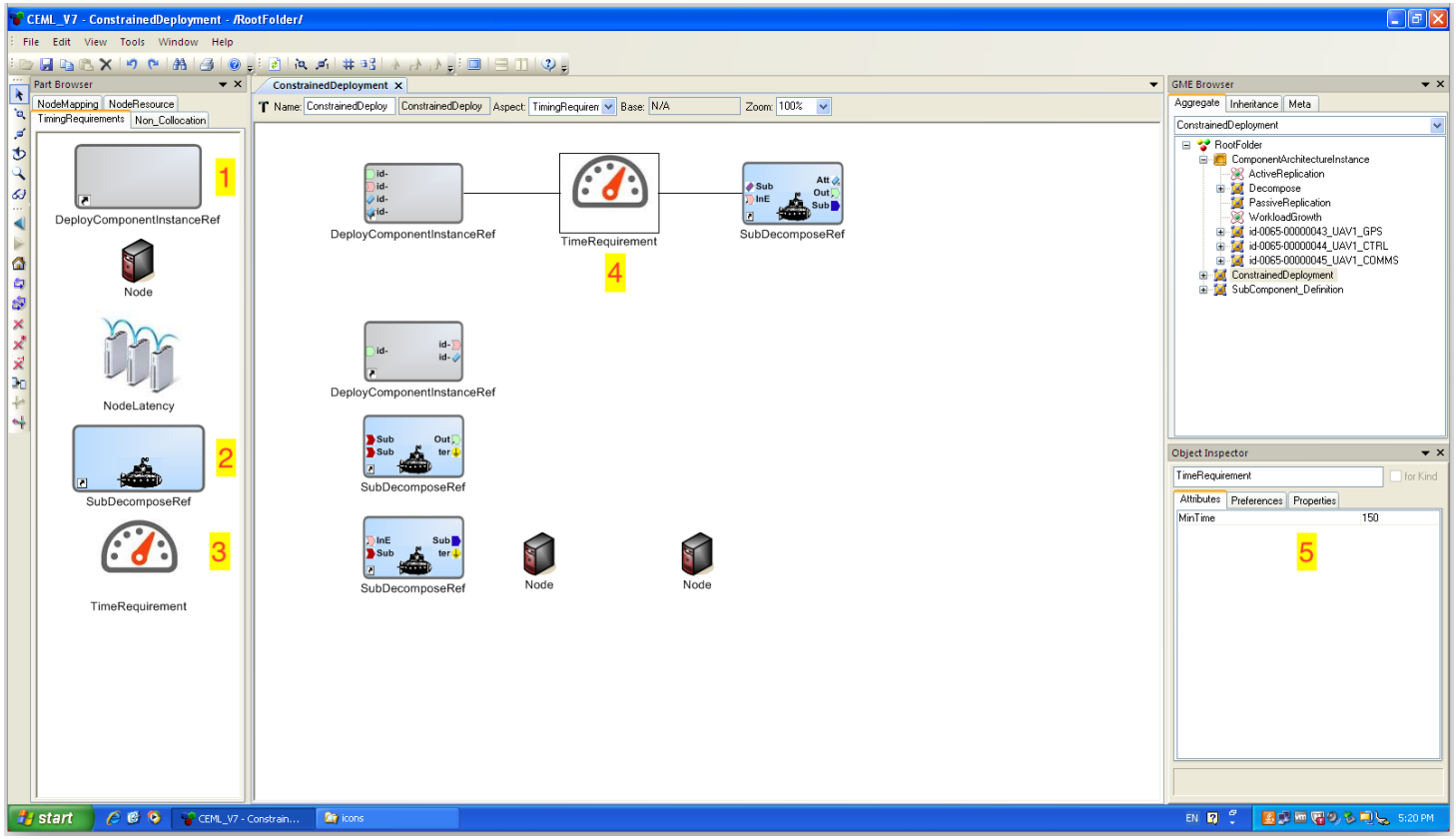

Figure 8.9: CEML modelling environment showing temporal performance constraint modelling

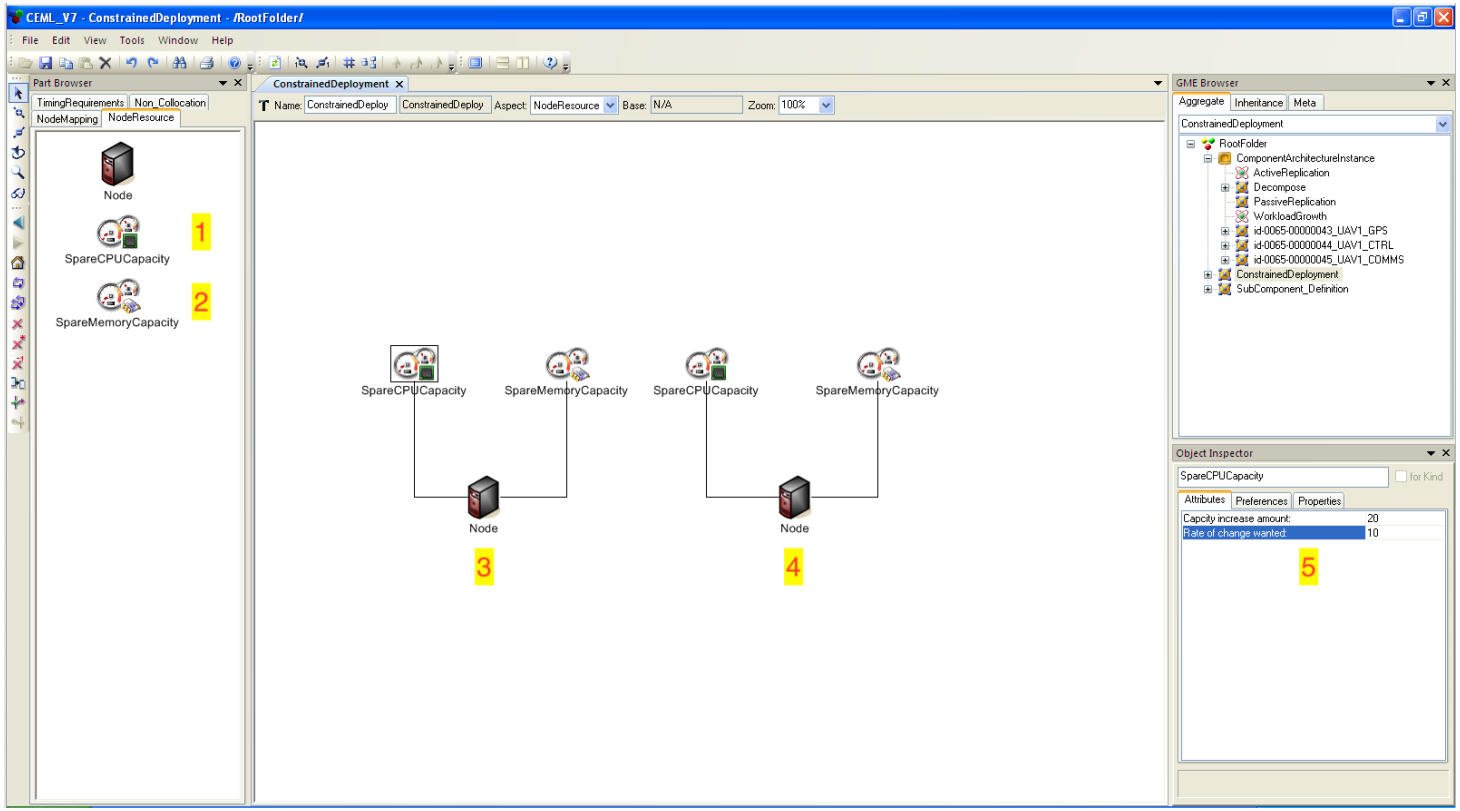

Figure 8.10: CEML modelling environment showing computing node resource capacity constraint modelling

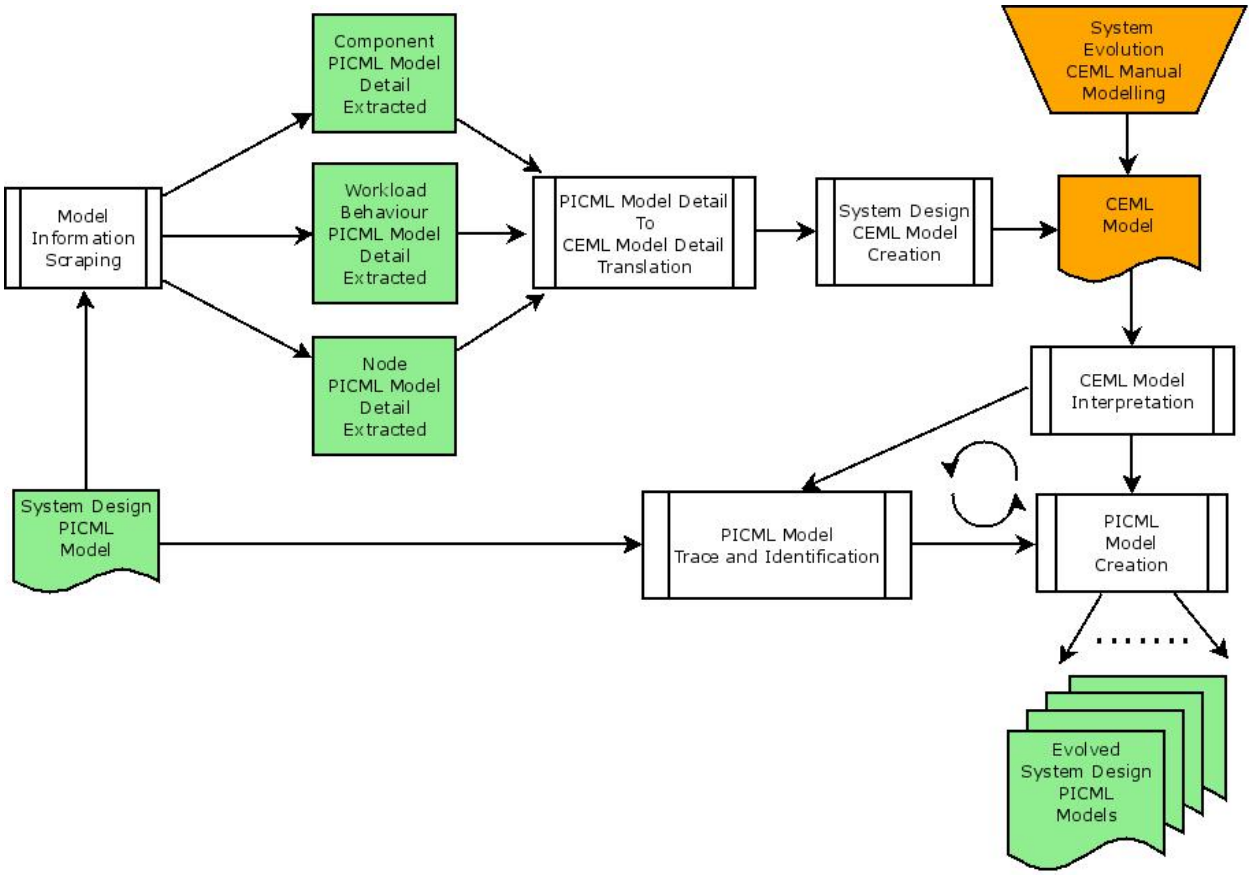

Figure 8.11: System design evolution framework execution flow

With the new, evolved system designs identified, the framework then uses the original PICML system design model to create new, evolved PICML system design models ready for deployment within the DIG framework. The final number of new PICML system design models constructed is based on the different combinations of evolved architectures (i.e., decomposition and replication), the different arrangements of particular evolved architectures (i.e., increasing the number of replication components up to the maximum) and number of run-time configurations considered for particular evolved architectures (i.e., increasing the workload up to the maximum evolved workload).

Figure 8.12 shows one of the evolved PICML UAV system design models constructed out of the CEML interpretation process from the architecture evolution definitions within the UAV CEML model. In this case, the evolved system design has all three of the defined architecture evolutions combined into the PICML single design model. This includes the active replication of the original *id-0065-00000043_UAV1_GPS* component (Position Markers 1-5), the decomposition of the original *id-0065-00000044_UAV1_CTRL* component (Position Markers 6-8) and the use of passive replications configured in a primary/shadow arrangement on the original *id-0065-00000045_UAV1_COMMS* component (Position Marker 9-12).

We can also see in the main modelling window all the newly-created component instances, along with the required connections. For the active replication, the original *GPS* component (Position Marker 1) remains executing, along with four new copies of it (Position Markers 2-5), also executing. Furthermore, all

connections for both inputs and outputs interface ports on all the components are connected and communicating.

In the case of passive replication, the model shown is in the primary/shadow replication configuration. In this configuration, the original *COMMS* (Position Marker 9) component remains executing along with three copies (Position Markers 10-12) executing in shadow mode. The input and output interfaces are connected for the primary original *COMMS*, while the shadow components (Position Markers 10-12) only have their input interface ports connected. The output interface ports of the shadow components (Position Markers 10-12) are not connected (Position Markers 20-22). Furthermore, to allow for state information to be shared from the primary component to the shadow components, the checkpoint interface exists. This interface is added to the original *COMMS* (Position Marker 16) along with modifying its other modelling artefacts, as well as adding the newly-created shadow component copies (Position Markers 17-19).

With each new model creation or existing model modification, all required modelling artefacts associated with component implementation and component architecture (Position Marker 23), component structure and behaviour (Position Marker 24) and message structure and interface definition (Position Marker 25) are also created, or modifications are made to existing artefacts. Furthermore, when decomposition occurs, the original component associated with the decomposition is removed within the system component architecture model (as depicted in Figure 8.12).

In the case of the UAV system design, CEML modelling interpretation process

would result in the construction of a number of new PICML system design models, based on the 24 combinations of the 'GPS' and 'COMMS' component replications, combined with the result of the decomposed component representation (future research could consider stepping through the number of sub-components generated up to the maximum). This number is then increased, based on the run-time configurations for the workloads defined (i.e. a 10% step increase for each design would result in an additional 10 designs for the workload associated with the 'GPS' component), and possibly increased further with step growth for workloads associated with the sub-components. Furthermore, the number could be increased again if there is a requirement to explore the other configuration options for passive replication modelling, such as 'check-pointing' or 'passive replication'.

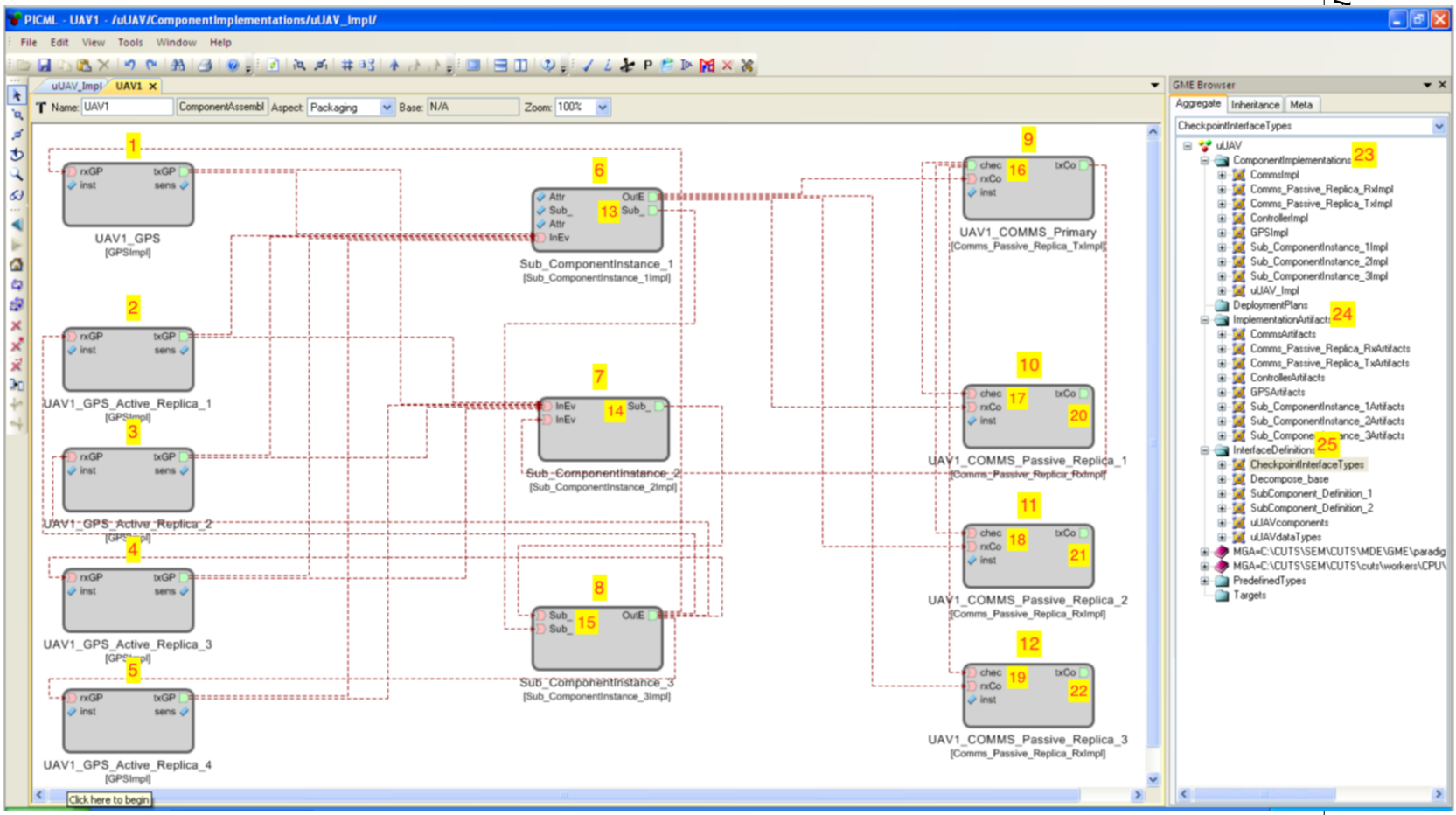

Figure 8.12: Example of one of the evolved UAV software system models

8.2 Modelling Framework

The aim of this modelling framework is to augment the analytical capability found within the SEM modelling environment with the new DSML and modelling mechanisms to explore system design evolution areas of interest. Through a series of model information extractions, translations and transformations, the overall capability enables the creation of new system design models that can then be fed into a SEM environment for execution and measurement-based analysis. This would include the utilisation of the newly created SEM environment deployment optimisation capability, also introduced with this thesis. The approach introduced here is built on the SEM environment created out of the research by Falkner et al., 2013.

The performance evaluation framework for the SEM environment utilised for this research, known as the DIG framework, is depicted in Figure 8.13. Because this SEM environment served as the initial research and development foundation that lead to the research and development of the MEDEA SEM environment (Falkner et al., 2018), we see the same key pillars of capability: namely, the building of models to represent system design and performance requirements, the execution of those models onto representative computing environments, and extraction of instrumented data to enable measurement-based analysis.

An important aspect of this work was that all architectural changes and newly created system models resulting from the evolution exploration capability were anchored to the original system design modelling paradigm. While different model

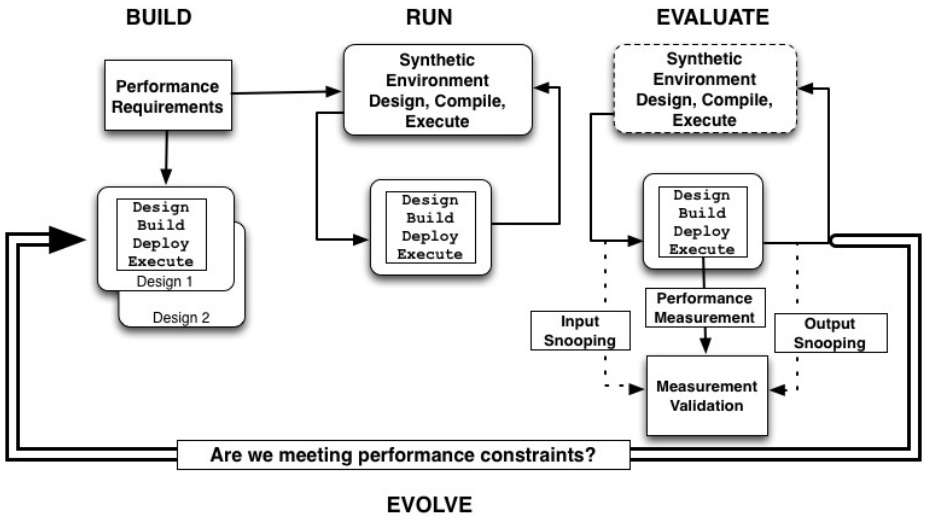

Figure 8.13: Overall DIG framework performance evaluation

paradigms and information are used, the entry and exit points from this new capability needed to be consistent and coherent with the original SEM environment artefacts. To achieve this our design needs were:

- To minimise complexities by maintaining the use of abstract MDD approaches where possible,
- To utilise the same code artefacts from the extant system design model to construct the evolved system architectures, behaviours and workloads,
- To translate high-level system requirements or constraints into a pictorial or text-based models.

Figure 8.14 shows a breakdown of the main components of the system design evolution performance prediction framework. The entry point into the framework is the extraction of extant software design information and definitions captured

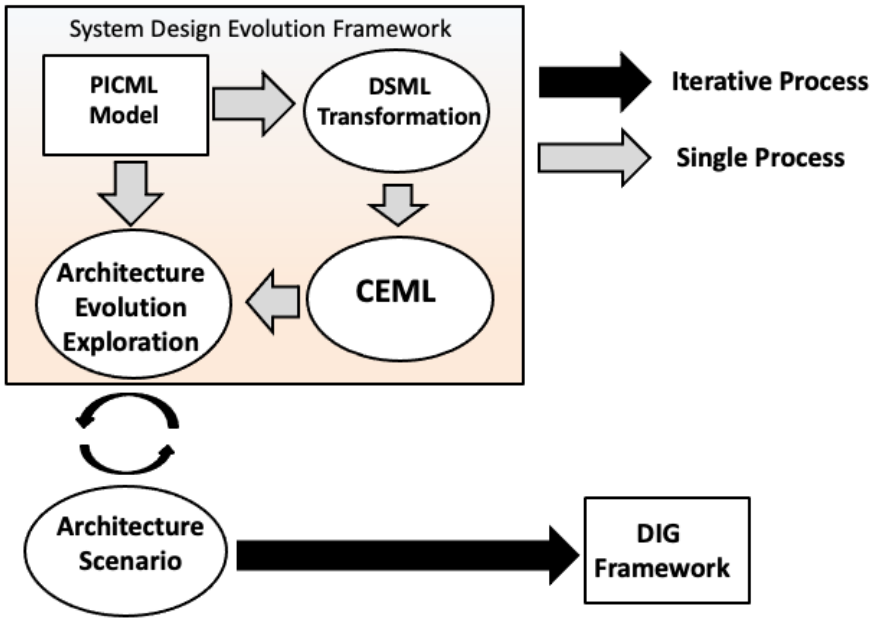

Figure 8.14: System design evolution performance prediction framework

using a DSML known as PICML (Platform Independent Component Modeling Language). This is followed by a DSML to DSML model translation to enable insertion into a CEML DSML model. The CEML model is then updated and used to search possible architecture evolution paths and the creation of new system design PICML models. Finally, the new PICML system design models are executed within the DIG framework.

The main elements of the framework are:

- **The PICML Model**: is a DSML constructed within a modelling environment that allows for creation of domain-specific modelling and program syn-

thesis environments, known as the Generic Modelling Environment (GME)[3]. This PICML DSML is used by the CoSMIC (Gokhale et al., 2003) / CUTS (Slaby et al., 2006) environment to develop models that describe a system's architecture, its deployment alongside description of component behaviour and resource consumption,

- **DSML Transformation**: is a Python script responsible for extracting, transforming and translating the appropriate parts of the PICML system design model to a CEML-based model,
- **CEML**: is a DSML developed within GME to allow for descriptions of areas of the system design needing to be explored for possible evolution paths. The CEML model itself, which runs within GME, guides exploration of ways to evolve a system design through pictorial modelling for such elements as architecture growth, deployment, resource requirements and deployment constraints,
- **Architecture Evolution Exploration**: is a Python script that uses the information and definitions found within a CEML model to search for possible system architecture change options and resulting evolved system designs.
- **Architecture Scenario**: the deployment of a particular evolved system design option onto the computing resource environment to be tested,
- **DIG Framework**: The model-driven SEM system design performance evaluation framework.

As depicted in Figure 8.14, four of the seven components within the frame-

[3]https://www.isis.vanderbilt.edu/Projects/gme/

work make up the System Design Evolution Framework. The capability of this framework is aimed to extract the smallest subset of extant model details required to produce a system design representation with CEML. It then allows the new CEML model to be augmented with the system architecture evolution characteristics wanting to be investigated, and then enables the identification and creation of new architectures for the extant system design, which leads to the creation of new evolved system design models.

In this initial research effort, the creation of evolved system designs was based on creating all permutations of architecture changes based on the CEML model. Future research efforts would explore the use of evolutionary algorithms to search for most likely architecture evolution paths.

The execution flow of the System Design Evolution Framework is depicted in Figure 8.11. Below is a description of each element that makes up that execution:

- **Model Information Scraping**: this process traces through the extant system design PICML model for information relating to the software architecture and workloads associating with every software component found within the system design model. The information scrapped is a minimum set of model details to allow for a representation of all the software components, event ports within software components, workloads associated with the event ports, and finally the connection between software components and their event ports. It also extracts details relating to the run-time environment available for deployment. Furthermore, for every model detail extracted, the PICML ID is extracted to support the requirements of the PCML Model Trace and

Identification process later in the execution flow.

- **PICML Model Detail To CEML Model Detail Translation**: this process takes all the extracted model details in PICML form and translates them into the required format to enable them to be held within a CEML model.
- **System Design CEML Model Creation**: this process takes all the model information translated into the CEML format and places it into an XML file template that can be imported into the CEML modelling environment.
- **System Evolution CEML Manual Modelling**: using the CEML modelling environment created through the use of GME and the CEML meta-model, the imported CEML extant system design model (the created XML file) can be annotated with modelling constructs to represent architecture evolution characteristics for particular parts of the system design.
- **CEML Model Interpretation**: taking the completed CEML model with system architecture evolution details this process identifies the number and type of changes to occur for each area of interest in the system architecture and then determines permutations of new system designs requiring creation.
- **PICML Model Trace and Identification**: working through all the architecture changes for each system design change permutation this process uses the PICML ID information to identify areas of the PICML extant model requiring change. Once these are found, a trace through the entire PICML model occurs to identify other, associated modelling elements requiring change. With the full modelling thread mapped out the required modelling changes occur to represent the particular system architecture change. All

change threads are then combined to represent each system design change permutation. The complete set of system design change permutations are then stored for use later in the execution flow of the PICML Model Creation process.

- **PICML Model Creation**: taking the set of system design change permutations and a copy of the extant PICML system design model, this process creates a new PICML model representing a particular architecture's set of changes and the resulting evolved system design model. This new PICML model is then ready for deployment within the SEM run-time environment.

8.3 Transition to MEDEA SEM Environment Design

While this body of research was built around the DIG framework, the overall research of which this thesis was a part of had deprecated this framework and developed the new MEDEA SEM environment. As a result, the integration software associated with the DIG framework and the DSML translation became obsolete, and a new design and second integration effort was required.

Figure 8.15 shows the new design execution flow of the overall system design evolution performance prediction framework, which builds on the MEDEA execution flow (Figure 4.9) including the deployment optimisation capability. For this new design the integration point occurs after the MEDEA modelling phase, with the delivery of the MEDEA system design model to the 'Model Information

Scraping' process. Furthermore, it can be seen that the original system design evolution framework and execution flow largely remains unmodified (indicated by the purple colouring). The remaining needs were to address the modification requirements of replacing code associated with the PICML DSML translation and modelling environment, integration with code needed for MEDEA DSML translation, and modelling environment integration.

The new design depicted in Figure 8.15 also shows the two new components (indicated by the blue colouring) addressing a capability gap to package models and provide an automatic single delivery point for deployment optimisation. This would build on the process introduced for delivering model files out of the deployment optimisation phase to the Jenkins Continuous Integration Server via the batching interface, which was developed as part of this thesis.

The scope of this thesis does not include this second development and integration effort. As a result, the case study section presented below will walk through the modelling and performance prediction approach with manual intervention to account for gaps within the overall integrated framework.

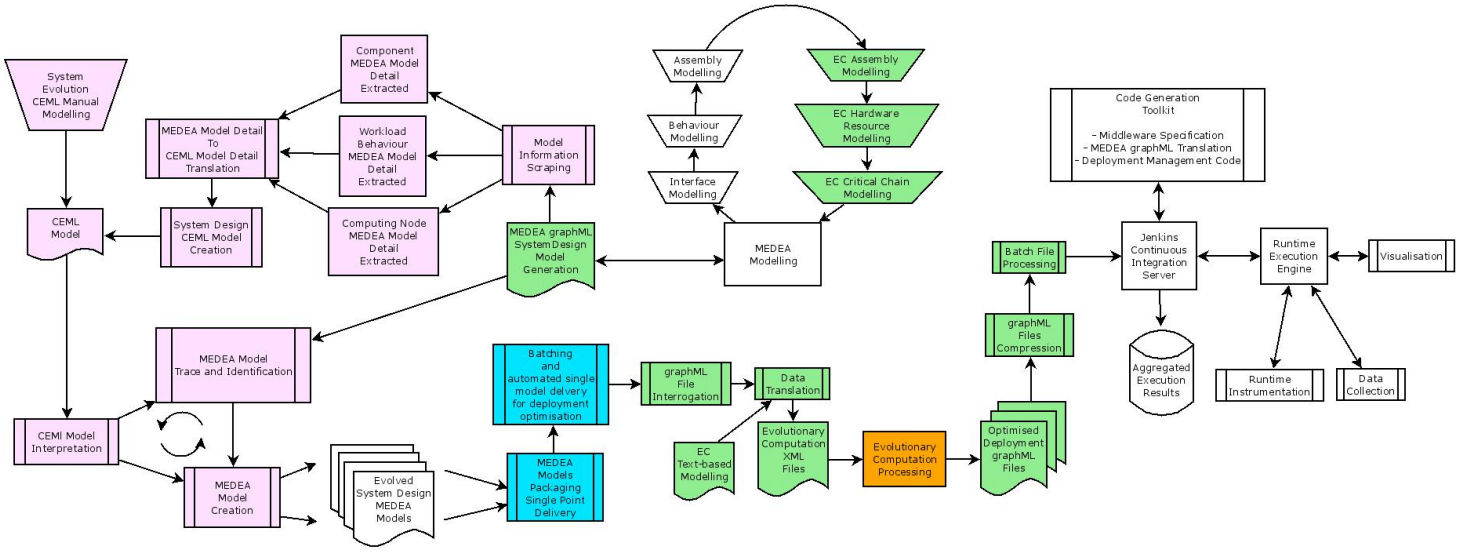

Figure 8.15: Overall system design evolution performance prediction framework integrated with the MEDEA deployment optimisation environment

8.4 Case Study

The case study described here is based on another example of an undersea sensor system. Due to the MEDEA DSML translation and integration framework not being available, the MEDEA system design model constructed (Figure 8.16) was translated manually to allow for modelling within the CEML modelling environment. This included the definition of the software system design and available computing nodes within the undersea sensor system network (Figure 7.1). This manual translation also included the modelling information from the CEML modelling environment back to the MEDEA modelling environment to enable deployment optimisation, execution and measurement-based performance analysis.

8.4.1 System Design Evolution Definition

Figure 8.17 shows the CEML component architecture representation of the undersea software system developed within the MEDEA modelling environment for this case study. As detailed above, the translation process from MEDEA to the CEML only considered the components themselves, their interfaces and the connections between the component interfaces.

Figure 8.16: CEML DSML case study: undersea software system design

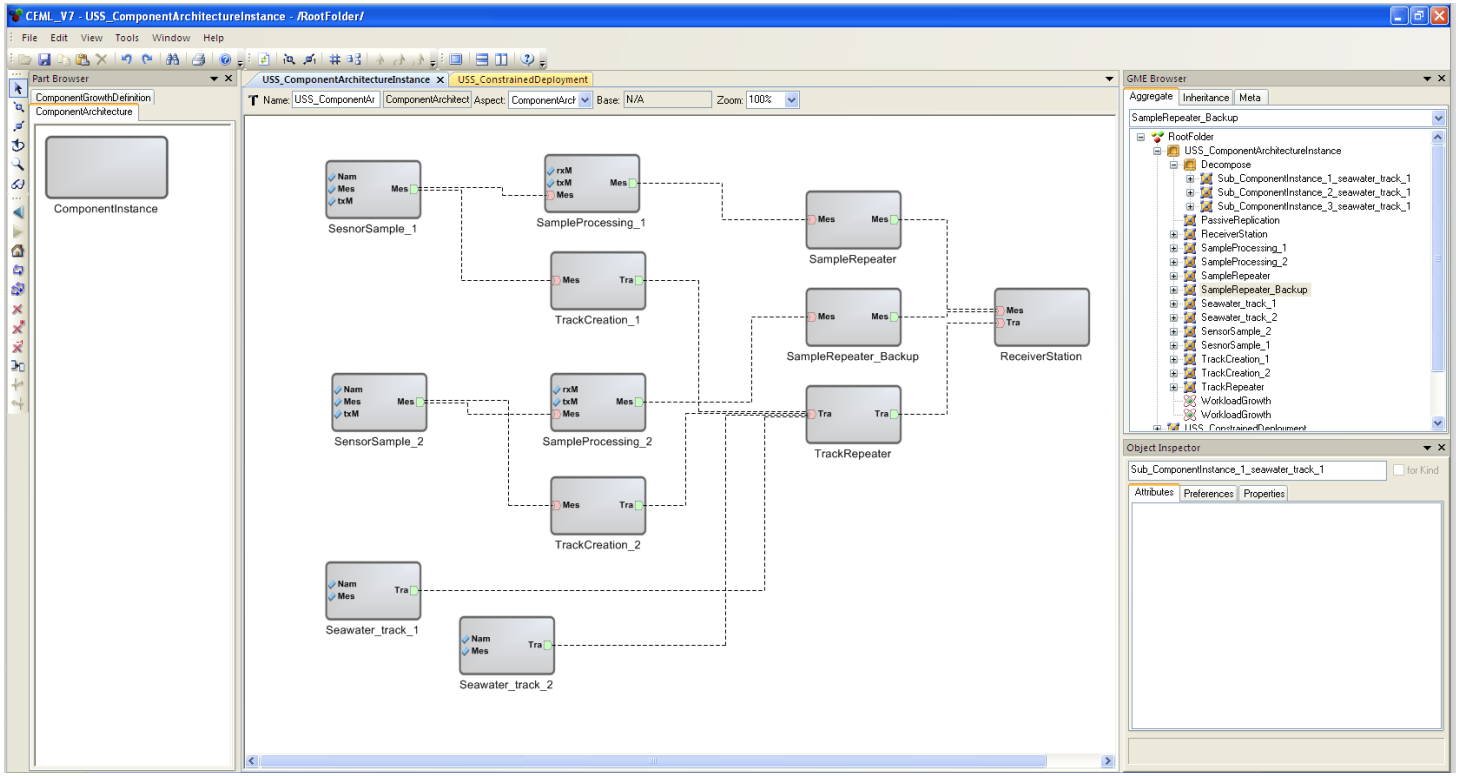

Figure 8.17: Undersea software system: component architecture within the CEML modelling environment

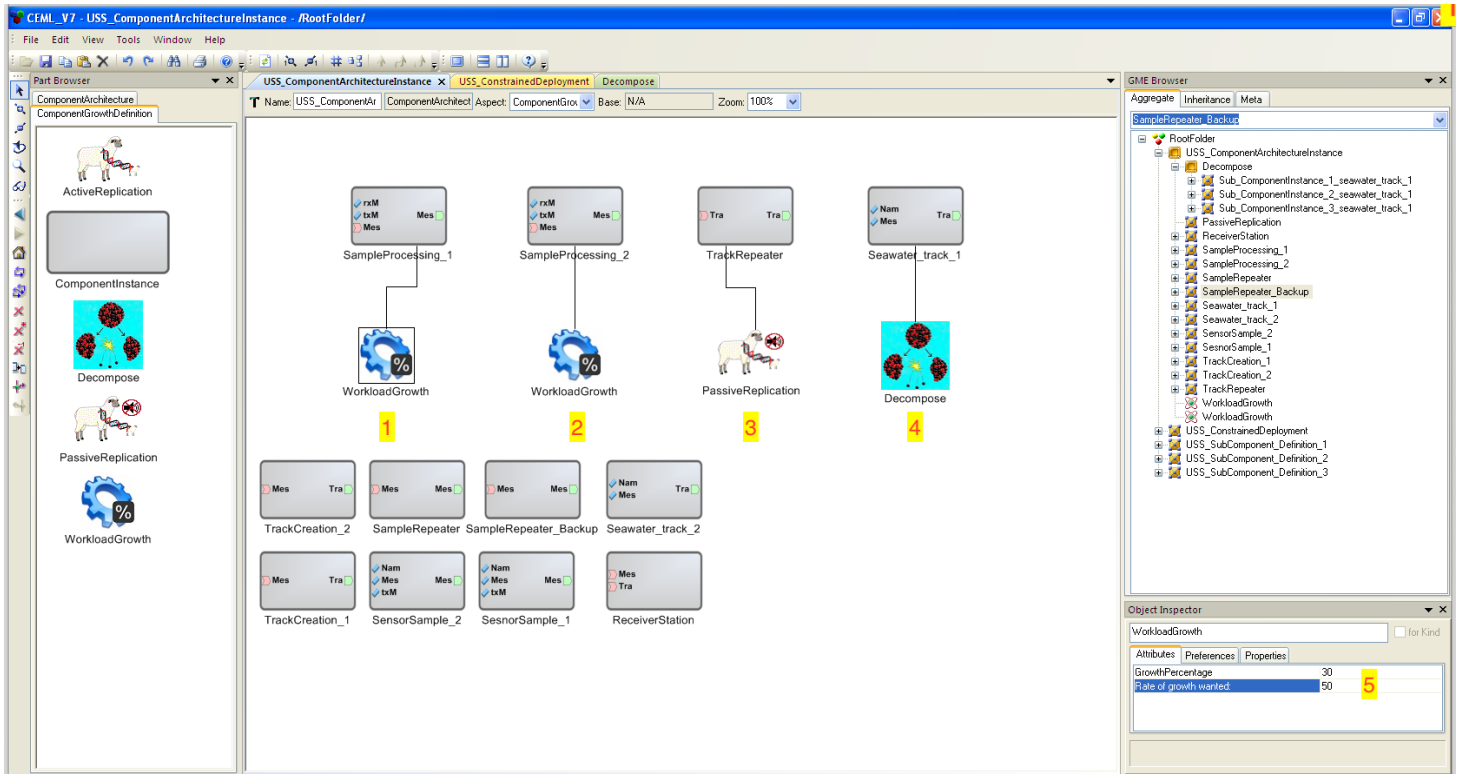

Figure 8.18: Undersea software system: component growth modelling within the CEML modelling environment

Using this case study, the CEML environment will demonstrate its application for investigating software system evolution, which includes software decomposition, introduction of redundancy and improved temporal performance. It will also demonstrate its use when investigating design evolution and emergent properties. In this scenario, it will also be accounting for battery performance degradation due to age.

Figure 8.18 shows CEML component growth modelling applied to the undersea sensor system software components. The architectural evolution definitions depicted are:

- **WorkloadGrowth (Position Markers 1 and 2:** workload increases are wanted within the two software components *SampleProcessing_1* and *SampleProcessing_2*, the increases being a 30% increase on current workload levels. Furthermore, the exploration considers the increase over two steps of a 15% increase and then the full 30% increase (Position Marker 5),
- **PassiveReplication (Position Marker 3):** the *TrackRepeater* software component requires improved redundancy, so passive replication is being applied to it. The type of passive replication explored was Primary/Shadow, therefore the input ports of each replica will be active and a checkpoint interface will be required (refer to Figure G.3 for modelling of this). The number of replicas required was two,
- **Decompose (Position Marker 4):** the *Seawater_track_1* software component was transitioned from a monolithic software application design to a distributed application through the development of a number of sub-components.

The investigation was required to explore the impact of a different number of sub-components and required connections, as well as new, distributed workloads for the new sub-components. It also included a latency constraint that was applied to the original output interface.

Figure 8.19 shows the decomposition modelling for the *Seawater_track_1* software component within the CEML modelling environment. In the main modelling window, we can see the decomposition wanting to explore the option of breaking up the monolithic software component into three sub-components. It also defines new connections between these sub-components, along with the original output interface from the monolithic software component (Position Marker 3). Appendix G shows the definition of the three sub-component behaviours, where *Sub_ComponentInstance_1_seawater_track_1* has a 20% portion of the total original software component workload, with four increased steps of 25% (Figure G.5), *Sub_ComponentInstance_2_seawater_track_1* has a 40% portion with no increased steps (Figure G.6), and *Sub_ComponentInstance_3_seawater_track_1* has a 50% portion with four increase steps of 25% (Figure G.7).

The last part of the CEML evolution modelling for the undersea software system defines the mandate deployment, temporal performance and resource capacity constraints applied to the evolved software system design and deployment to the undersea sensor system network resources.

Figure 8.20 shows the mandate deployment constraint modelling for the *Seawater_Sensor_1* node within the undersea sensor system network. In this case, the newly-developed component *Sub_ComponentInstance_1_seawater_track_1*

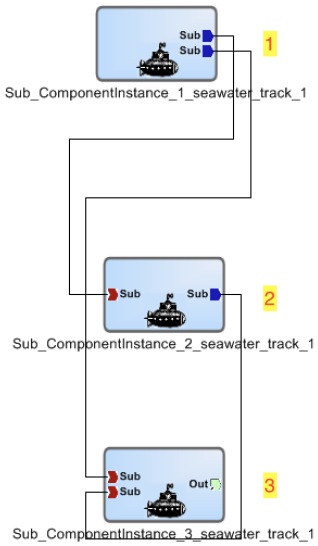

Figure 8.19: Undersea software system component decomposition modelling within the CEML modelling environment

(Position Marker 3) has been associated with the mandate deployment group model entity (Position Marker 2), which is tied to the *Seawater_Sensor_1* computing node (Position Marker 3). In addition to the new mandated deployment constraints defined, mandated deployment constraints defined in the original MEDEA are also inherited, and can be modified as required.

Following mandate deployment constraint modelling, the temporal performance constraints are defined. Figure 8.21 shows a temporal performance constraint modelled between the newly developed *Sub_ComponentInstance_3_seawater_track_1* component (Position Marker 1) and the connection to the *SampleProcessing_1* component (Position Marker 3). Furthermore, as this connection is associated with a temporal performance constraint modelled within the extant MEDEA design

model, the maximum latency definition within the CEML model (Position Marker 4) leads to a further improvement (decrease) of the maximum latency performance for the connection.

The no-collocation constraint defined for the undersea sensor system relates to the redundancy improvement introduced with the component architecture growth modelling. In this case (Figure 8.22), the passive replication (Position Marker 2) associated with the *TrackRepeater* component has been modelled (Position Marker 1) to ensure the replicas created will not be collocated when deployed onto the undersea sensor system network.

As detailed above, this case study for the undersea sensor system considers a scenario where battery performance degradation has occurred due to age. In this case (Figure 8.23), the degradation is with computing nodes *Seawater_Sensor_2* and *Sea_Buoy*. To account for this power supply performance reduction the levels of spare CPU capacity have been increased for both nodes (Position Markers 1 and 2) to account for an additional increase of 20%. Furthermore, the investigation will consider four increased steps up to the 20% total point (Position Marker 3).

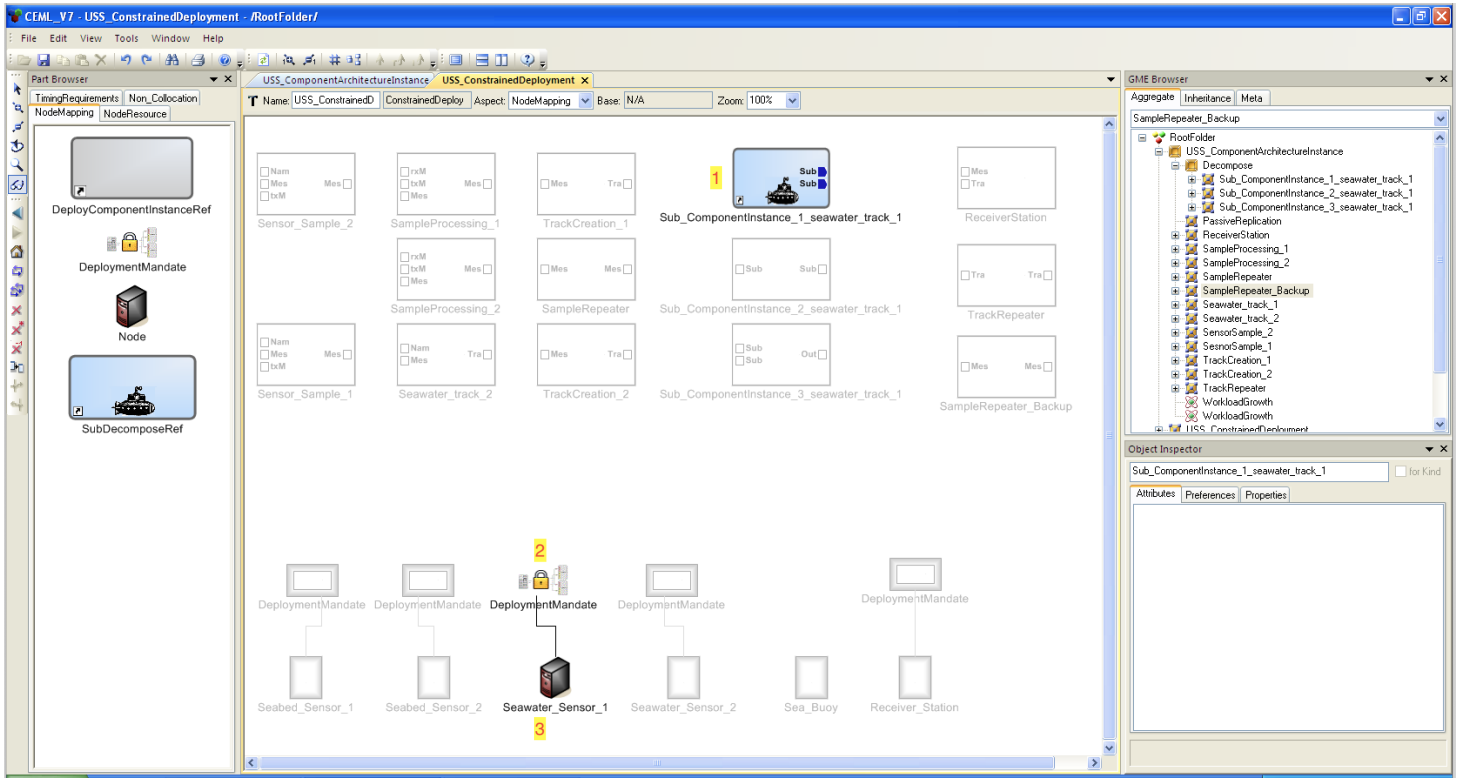

Figure 8.20: Undersea software system mandate deployment modelling within the CEML modelling environment

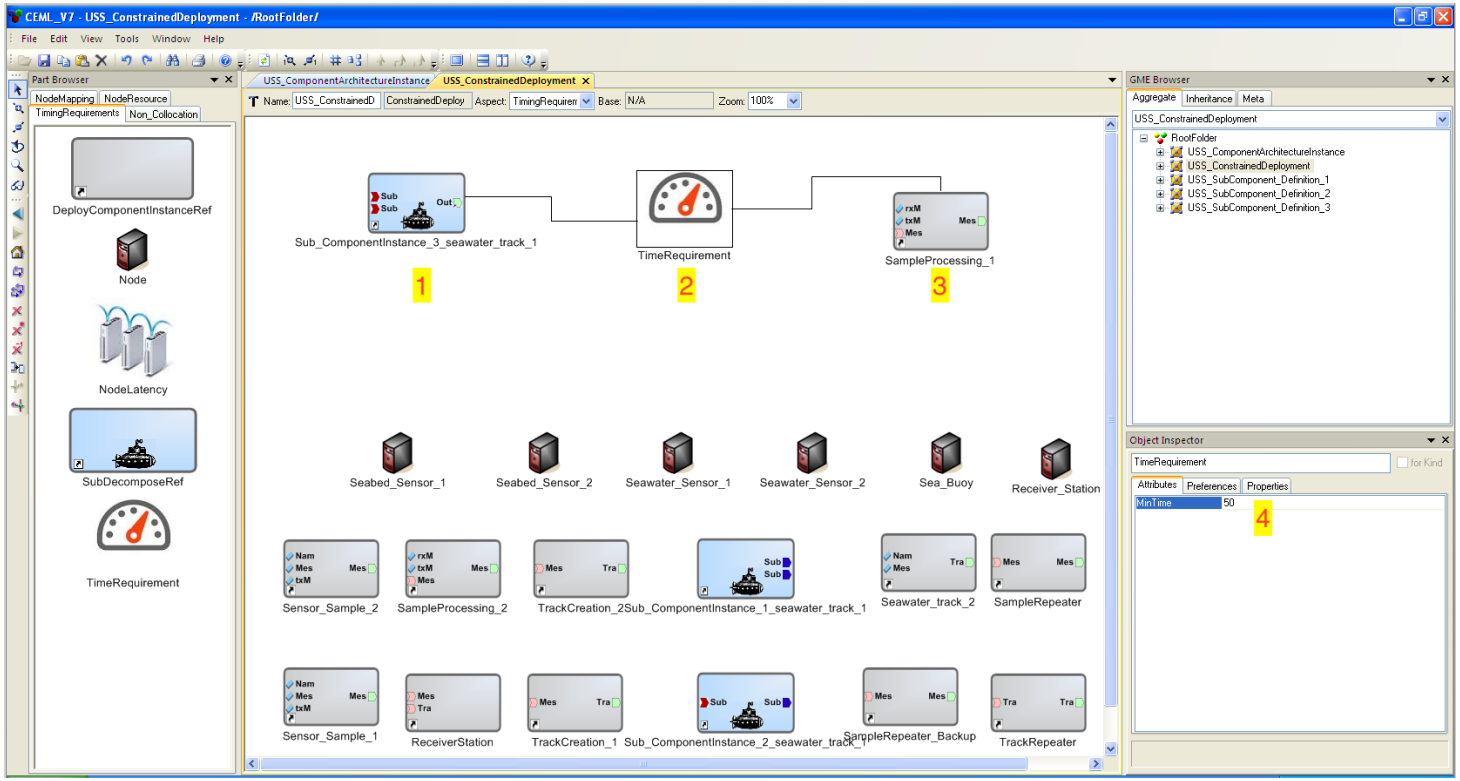

Figure 8.21: Undersea software system temporal performance modelling within the CEML modelling environment

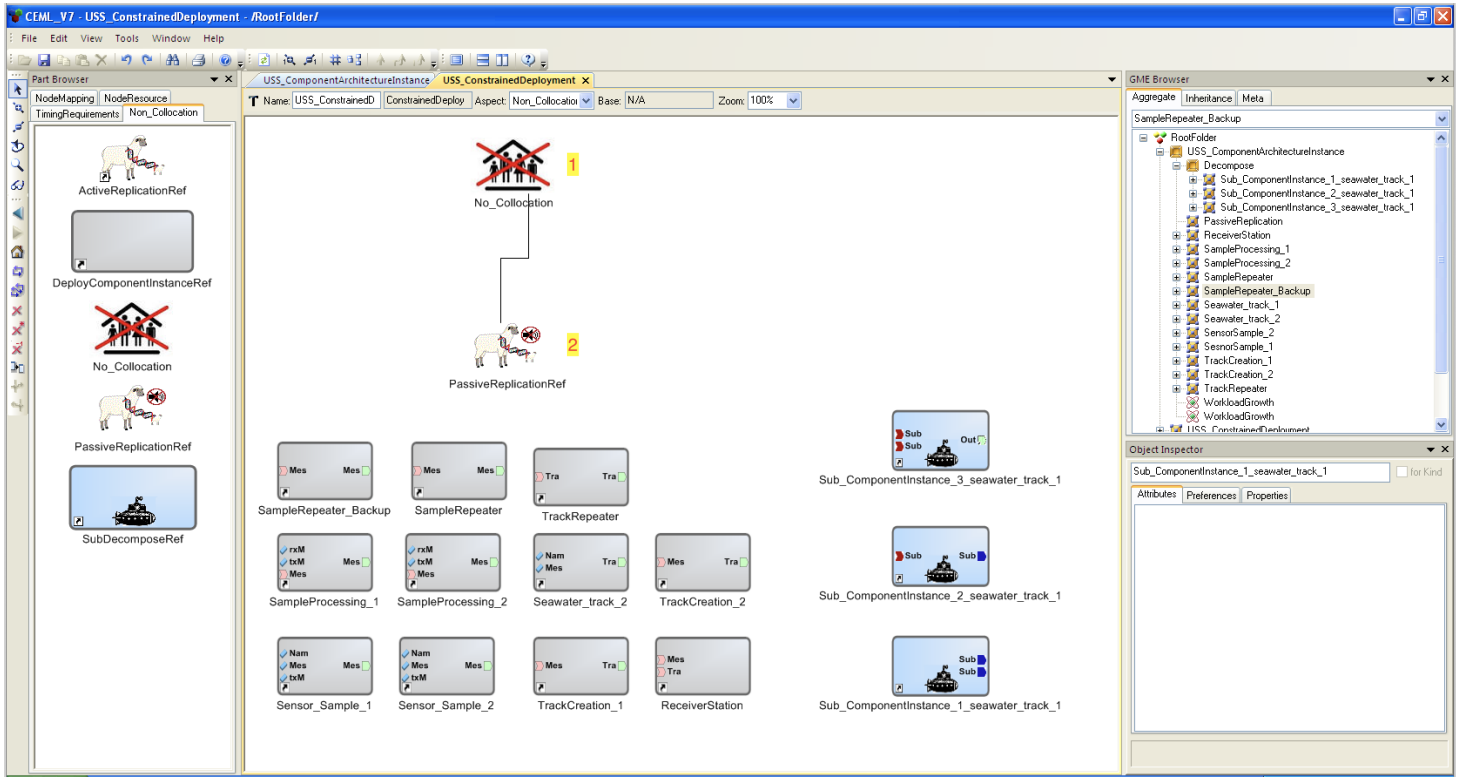

Figure 8.22: Undersea software system no-collocation modelling within the CEML modelling environment

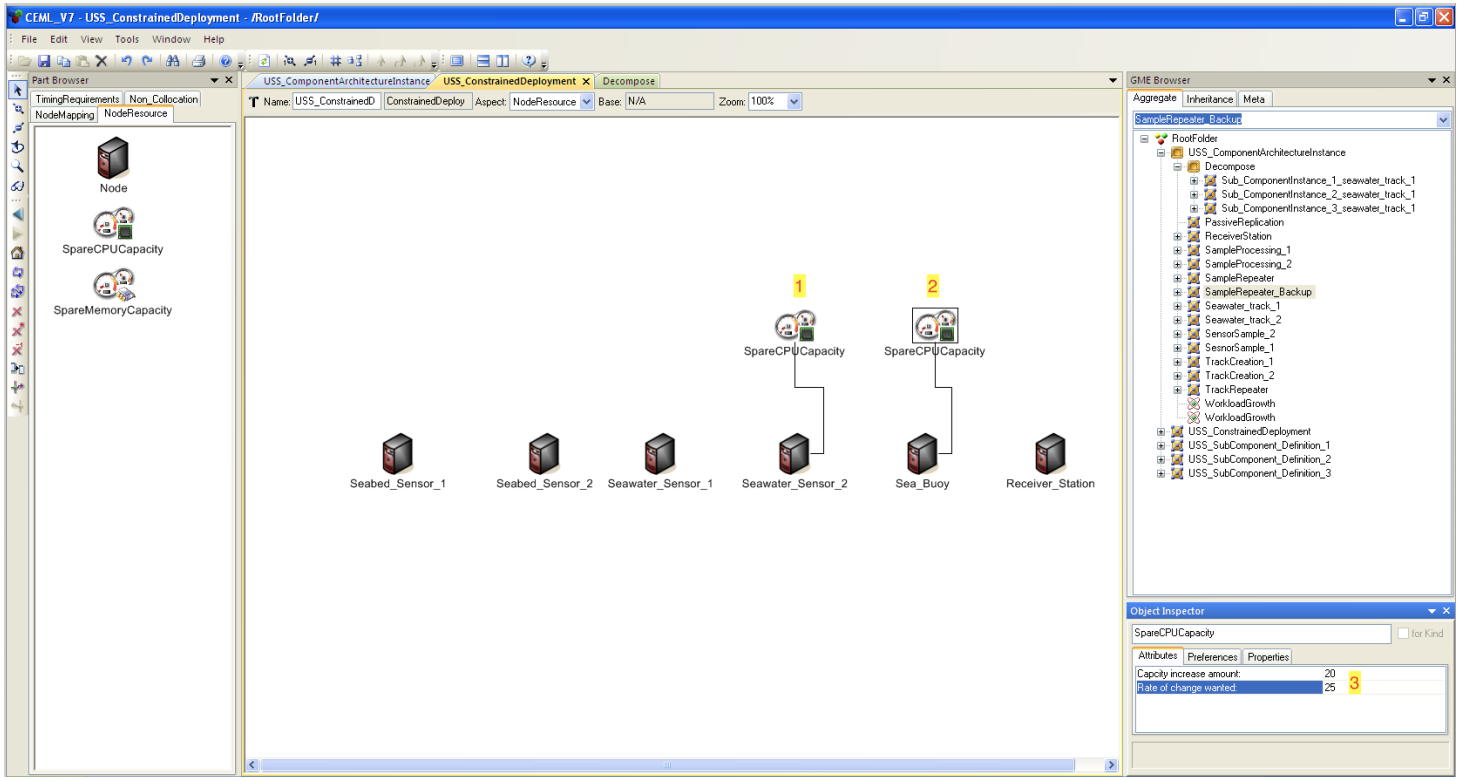

Figure 8.23: Undersea software system CPU spare capacity modelling within the CEML modelling environment

With the completion of the CEML definitions, the CEML interpretation process is executed to develop all the permutations of evolved system designs and configurations. An example of one of the evolved MEDEA undersea system designs can be seen in Figure 8.24. Position Markers 1, 2 and 3 indicate the replication addition where there is the primary function (Position Marker 1) and two passive shadows (Position Marker 2 and 3) are keeping in synchronisation with the primary software component but are not active. Meanwhile, Position Markers 4, 5 and 6 indicate the new decomposed software component where the original monolith has been broken into three new sub-components. To support the development of these new sub-component instances, the CEML interpretation process also creates the new software component definition. In this case, three new sub-component definitions are produced and one of these definitions can be seen in Figure 8.25.

With the introduction of the integration between the CEML and MEDEA environments, all the designs and configurations can then be bundled and delivered to the MEDEA for deployment within its workflow. This would also utilise the new unpacking and execution capability and the deployment optimisation capability also introduced with this thesis. The result of this project is that system designers will not only gain the ability to explore optimal deployment of a known software system onto the system's computing infrastructure, but also to explore how optimal deployment of an evolving software system performs when it is deployed onto the same computing infrastructure.

Figure 8.24: An example of one evolved undersea software system design

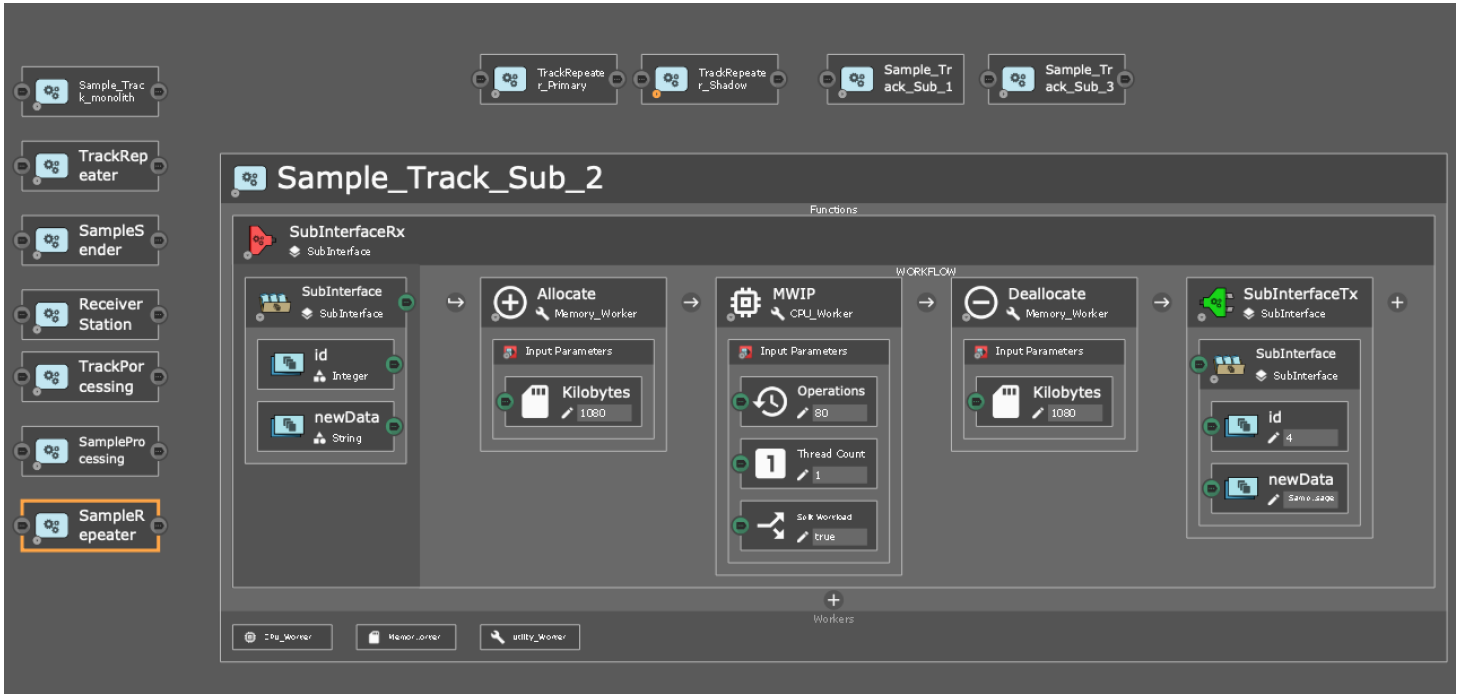

Figure 8.25: Evolved undersea software system design new component definitions

In the case of an undersea sensor system, this would enable investigations into how best to deploy the software as it evolves onto the existing computing environment, how to gain insight into performance impact, and how to evaluate and change requirements and constraints, based on different performance expectations. Furthermore, it would also enable the exploration of how to feed in the effect of battery capacity loss (through computing resource evolution modelling), or communication performance constraints (through temporal performance evolution modelling) from environmental changes.

The introduction of the CEML DSML did, in fact, enable a reduction in complexity and effort in defining the various characteristics and behaviours needed to capture the system evolution. It also showed potential as a guide to the modeller through all the modelling requirements when generating complete definitions. However, improvements in the level of abstraction are still needed to reduced the amount of text-based annotations, as well as the number of modelling entities needed to complete the definitions. This abstraction improvement would also help to improve the scope of areas of the system design to be considered as part of the evolution modelling.

9. Conclusions and Future Work

Systems of systems (SoS) are found across many domains, including defence, power networks, space and avionics, and automotive systems. SoS are a collection of systems that work together to achieve a common purpose, where the system's desired emergent behaviour results from the integration of independently operating and managed heterogeneous systems. SoS are never viewed as achieving a finalised state: they evolve in response to continually changing operational requirements and have extremely long life cycles. In addition, mission-critical SoS are rarely built from scratch and so they integrate legacy systems. As a consequence, significant limitations exist for integration and operational deployment, resulting in potentially unexpected and undesirable overall system performances at times.

The literature presented in this thesis showed that while an understanding of both functional and non-functional aspects of the SoS architecture is important, non-functional aspects are of greater concern for resource-constrained platforms, as there is little to no ability to improve processing capabilities once the host systems are developed. With strict budget allocations for space, weight and power (SWaP) for various systems installed on many of these SoS host platforms, any early insight

into the performance of the SoS from its corresponding deployment becomes crucial in preventing an imbalance between resource availability and demand, which can otherwise result in performance shortfalls and potentially expensive rebuilds. The literature also showed there are challenges in SoS development to minimise the risk of architectural change late in the development cycle, and the associated high costs. As a result, modelling methodologies emerge as a crucial tools in prevention of these late changes by offering key insights into deployment and non-functional properties early in the development and integration cycle (for the initial development) or subsequent upgrade cycles.

The literature also showed that early insight cannot be achieved through traditional methodologies, as they focus first on functional aspects of the system design and leave the non-functional aspects until the final phases of the development life cycle. The literature showed that any system design modelling needs to capture system characteristics and behaviours, both static in architectural design and dynamic in system execution. In addition, by allowing for mapping of models onto real computing platforms for evaluation of non-functional requirements and constraint satisfaction, significant benefit could be gained from earlier identification of design risks associated with constrained SoS through measurement-based analysis. This included low-level insights into behaviour and performances of the ever-increasing number of COTS elements that make up a constrained SoS design.

Our literature review found pockets of performance prediction modelling research that focus on measurement-based techniques and deliver the level of fidelity required for early insight into COTS-based constrained SoS design risks. Lead-

ing the way into measurement-based techniques was research known as System Execution Modelling (SEM). Using a technique of deploying abstract application business logic models and then executing them on actual real computing infrastructure, key insights into behaviours and performances, as well as integration, were produced through post-execution measurement analysis. However, the literature also showed the SEM technique was predominantly focused on the representation of system architecture and its workload. The SEM approach did not consider large scale searches associated with the deployment of the many SoS software elements, or an ability to optimise any such deployment around known requirements or constraints for the system development effort. It was also shown that an extension to the optimal software deployment search capability was to have the search consider the evolution of the software system. By introducing a technique that allows for the optimisation capability to account for system evolution characteristics in addition to the high-level requirements and constraints, optimal software deployment could be explored for evolved system design options.

The literature survey considered optimisation techniques to address the complexity of searching the software deployment problem space. Optimisation techniques can be found solving problems across many computing paradigms, including software deployment. However, these approaches considered deployment optimisation at the system or platform level and not the software component level. The literature also showed system evolution performance prediction techniques were either postmortem-based or largely abstract in nature, and not to the level of fidelity required for performance insights associated with COTS-based constrained SoS

design evolution.

In response to these gaps, this thesis introduced new research for modelling deployment of large scale, evolving COTS-based constrained SoS. This thesis introduced an approach consisting of a new SEM modelling approach, optimisation algorithms to enable software deployment optimisation at the required fidelity, new instrumentation methods, and a new modelling language for evolution characteristics definition and construction of evolved software system designs. The proposed approach was extensively verified and validated, and the experimental analysis evaluated a realistic case study with several scenarios being showcased. In fact, through the application of the case study, the new optimisation capability was shown to provide critical and unique insights into optimised design exploration for the various performance aspects, such as mandate deployment, power consumption and temporal performance.

Novel SEM modelling approach: In response to the identified capability gap for SEM environments to handle deployment of large scale SoS design models automatically, this thesis introduced a novel SEM modelling approach centred around system design critical chains (or threads) and non-functional requirement modelling. By introducing new model entities and modelling processes, the system modeller is capable of declaring desired behaviour and performances associated with that identified critical chain, as well as non-functional requirements for the broader software system and available computing nodes. Using these declarations, the software deployment can then be automatically guided (the second novel

contribution) through a more formal mechanism, based on high-level requirements and constraints.

As a result, system designers are easily provided with insight and correlation on low-level system performances and behaviours, as a result of high-level design drivers.

There were some limitations with the modelling method within the SEM environment as a result of the level of integration. This led to large usage of text-based attribute annotation. Also, the initial method developed only considered a single critical chain within the system design, but it would be reasonable to expect system modellers would want to explore various design critical chains or threads.

Novel software deployment optimization algorithms: The literature survey identified research gaps with large areas of deployment optimisation research only considering deployment optimisation at the system or platform level. Furthermore, for those cases where software component granularity was considered it was around the individual elements largely in isolation, rather than broader system aspects. In response to this, this thesis presented novel optimisation algorithms and frameworks to address this capability gap and to serve as the automated searching capability complementing the novel SEM modelling approach.

The testing and analysis presented in thesis showed the optimisation technique introduced operated and predicted correctly for each element of its algorithms. The results presented showed some performance shortfall, largely due to fidelity of the models developed (which can be improved with future research). They also showed

the overall performance and ability of the new optimisation approach correctly identified software component deployment options in response to deployment mandate requirements, resource utilisation constraints and temporal performance requirements. This was further conveyed with the application of the introduced technique to an undersea sensor network system design case study. This case study showed how the application provided critical and unique insights into optimal design exploration for the various performance aspects, such as mandate deployment, power consumption and temporal performance.

As with the novel SEM modelling approach, the introduced optimisation technique was limited to considering a single critical chain of the system design.

Novel instrumentation methods: The introduction of the novel modelling approach resulted in the need to construct a new SEM instrumentation approach. This approach allowed for various new run-time measurements and correlations of performances and events associated with the critical chain modelled with the broader instrumentation capability of the SEM environment. The approach centred around a data model implemented within the SEM model definitions and model instantiation, as well as post-execution data translation and correlation processes to enable analysis techniques to occur, such as the Pearson Correlation, Confidence Intervals, Mean Standard Deviation and Polynomial Fit.

With this data and subsequent analysis, it was verified that the Objective Function and Evolutionary Search algorithms performed as expected. It then confirmed the ability of the introduced capability to construct system designs that

deliver optimised performance for the known non-functional design aspects. It also served to support calibration to improve algorithm performance.

Novel domain language for expressing system evolution: From the literature it was identified that system evolution performance was mostly postmortem and forensic, while research forward was largely abstract in nature and not suited to the level of fidelity required for performance insight associated with mission-critical DRE SoS design evolution. As a result, this thesis introduced a new DMSL and interpretation and translation frameworks. These allowed SEM-based models to have system evolution characteristics annotated to enable design exploration and construction of new SEM-based models representing evolved system design options.

Known as CEML, this new language, along with the newly developed language translation and interpretation frameworks, presents a new method that allows system designers to gain insight into how best to utilise available computing resources when the software system evolves. This new language was demonstrated by working through an evolution of an undersea sensor system, where the evolution characteristics were defined and resulted in new, evolved design models being constructed through new evolution exploration processes and in readiness for execution within the MEDEA SEM environment. These processes would also include utilisation of the novel optimisation technique, also introduced with this thesis.

The research conducted within this thesis introduced four contributions to

the research community associated with modelling and predicting system design performance, including system evolution. In addition to realising these direct research outcomes, the thesis research also supported the broader SEM research program and its research outcomes and publications (Falkner et al., 2018, Falkner et al., 2013 & Falkner et al., 2013).

9.1 Future Work

Building on the successful demonstration of the new technique for searching for optimal software deployment of evolving systems with this thesis, several avenues for future work were identified.

Operational Environment Deployment: While the initial focus of the research for this thesis was around the design effort and insights for system designers, the methods introduced here could also be applicable to operational environment and dynamic changes to software deployment based on current operational conditions. High-level requirements and constraints could be used to adjust the software deployment in real-time, where the response could be based on such things as operational posture, failure or damage, or the desire to adjust and extend operational endurance. This would also consider testing methods to demonstrate robustness of the approach and assurance of the outcomes.

Multi-Critical Chain Optimisation: For the new SEM model approach and optimisation capability, improvements could be made to allow for multiple critical chains within a system design to be identified and reconciled to deliver software

deployment profiles and optimal performance from defined non-functional requirements.

Additional Processor Platforms:Improvements can be made to the optimisation algorithms to increase their scope in accounting for processing hardware beyond the CPU-based computing nodes. New capabilities can be added for consideration when deploying to GPU-based computing nodes, which would also introduce methods for applying new, non-functional requirements to the available resources. The optimisation algorithm could also be expanded to consider the use of shared infrastructure through the use of virtualization and technologies like Docker or microVMs, and how to consider the ability to dynamically change resources at the system deployment level, while still ensuring deployment and performance optimisation at the software component level.

CPU and Network Model Improvements: The models used for CPU and network utilisation within the optimisation process could benefit from having higher fidelity data added to them for interpolation purposes. As an alternative, the optimisation process could look into methods for directly applying instrumentation as part of the optimisation process. This could occur in the form of a test harness or possibly having the execution of models, instrumentation and optimisation within a single execution loop.

Full Environment Integration: Conduct development and integration of the CEML model framework with the MEDEA SEM environment (and the optimisation capability introduced with this thesis) could be used to develop a complete automated software system evolution deployment optimisation capability.

Bibliography

Abdelaziz, A.A., W Kadir, and A. Osman (2011). "Comparative analysis of software performance prediction approaches in context of component-based system". In: *International Journal of Computer Applications* 23.3, pp. 15–22.

Akdemir D.and Sanchez, J.I. and J. Jannink (2015). "Optimization of genomic selection training populations with a genetic algorithm". In: *Genetics Selection Evolution*. DOI: https://doi.org/10.1186/s12711-015-0116-6.

Akdur, D., V. Garousi, and O. Demirörs (2018). "A survey on modeling and model-driven engineering practices in the embedded software industry". In: *Journal of Systems Architecture* 91, pp. 62–82. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2018.09.007. URL: https://www.sciencedirect.com/science/article/pii/S1383762118302455.

Akoglu, H. (Aug. 2018). "User's guide to correlation coefficients". In: *Turkish Journal of Emergency Medicine* 18. DOI: 10.1016/j.tjem.2018.08.001.

Akyildiz, I.F., D. Pompili, and T. Melodia (July 2004). "Challenges for Efficient Communication in Underwater Acoustic Sensor Networks". In: *SIGBED Rev.* 1.2, 3–8. DOI: 10.1145/1121776.1121779. URL: https://doi.org/10.1145/1121776.1121779.

Amyot, D., H. Farah, and J. Roy (2006). "Evaluation of Development Tools for Domain-specific Modeling Languages". In: *Proceedings of the 5th International Conference on System Analysis and Modeling: Language Profiles*. Kaiserslautern, Germany: Springer-Verlag, pp. 183–197. DOI: 10.1007/11951148_12. URL: http://dx.doi.org/10.1007/11951148_12.

Armstrong, R. (Oct. 2008). "Symposium on Model-Based Systems Engineering Proceedings". In: *Symposium on Model-Based Systems Engineering*.

Awan, K., P. A. Shah, K. Iqbal, S. Gillani, W. Ahmad, and Y. Nam (Jan. 2019). "Underwater Wireless Sensor Networks: A Review of Recent Issues and Challenges". In: *Wireless Communications and Mobile Computing* 2019, pp. 1–20. DOI: 10.1155/2019/6470359.

Balasubramanian, K., J. Balasubramanian, J. Parsons, A.S. Gokhale, and D.C. Schmidt (2005). "A platform-independent component modeling language for distributed real-time and embedded systems". In: *11th IEEE Real Time and Embedded Technology and Applications Symposium*, pp. 190–199.

Balsamo, S., A. di Marco, P. Inverardi, and M. Simeoni (May 2004). "Model-based performance prediction in software development: a survey". In: *Software Engineering, IEEE Transactions on* 30.5, pp. 295 –310. ISSN: 0098-5589. DOI: 10.1109/TSE.2004.9.

Balsamo, S., M. Marzolla, and R. Mirandola (Sept. 2006). "Efficient Performance Models in Component-Based Software Engineering". In: *32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06)*, pp. 64–71. DOI: 10.1109/EUROMICRO.2006.34.

Barbierato, E., M. Gribaudo, and M Iacono (2011). "Defining Formalisms for Performance Evaluation With SIMTHESys". In: *Electronic Notes in Theoretical Computer Science* 275. Fifth International Workshop on the Practical Application of Stochastic Modelling (PASM), pp. 37 –51. ISSN: 1571-0661. DOI: http://dx.doi.org/10.1016/j.entcs.2011.09.004. URL: http://www.sciencedirect.com/science/article/pii/S1571066111000958.

Baxter, G., M. Frean, J. Noble, M. Rickerby, H. Smith, M. Visser, H. Melton, and E. Tempero (2006). "Understanding the Shape of Java Software". In: *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*. OOPSLA 06. Portland, Oregon, USA: ACM, pp. 397–412. ISBN: 1-59593-348-4. DOI: 10.1145/1167473.1167507. URL: http://doi.acm.org/10.1145/1167473.1167507.

Becker, S., L. Bulej, T. Bures, P. Hnětynka, L. Happe, J. Kofron, H. Koziolek, J. Kraft, R. Mirandola, J. Stammel, G. Tamburrelli, and M. Trifu (Sept. 2008). "Q-ImPrESS Project Deliverable D2.1: Service Architecture Meta-Model (SAMM)". In:

Becker, S., L. Grunske, R. Mirandola, and S. Overhage (2006). "Performance Prediction of Component-Based Systems". In: *Architecting Systems with Trustworthy Components*. Ed. by Ralf H. Reussner, Judith A. Stafford, and Clemens A. Szyperski. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 169–192. ISBN: 978-3-540-35833-6.

Becker, S., H. Koziolek, and R. Reussner (2007). "Model-Based Performance Prediction with the Palladio Component Model". In: *Proceedings of the 6th International Workshop on Software and Performance*. WOSP 07. Buenes Aires,

Argentina: ACM, pp. 54–65. ISBN: 1-59593-297-6. DOI: 10.1145/1216993.1217006. URL: http://doi.acm.org/10.1145/1216993.1217006.

Bertolino, A. and R. Mirandola (2003). "Towards Component-Based Software Performance Engineering". In:

Boudreau, M. (Apr. 2007). "Acoustic Rapid COTS Insertion: A Case Study in Modular Open Systems Approach for Spiral Development". In: *2007 IEEE International Conference on System of Systems Engineering*, pp. 1–6. DOI: 10.1109/SYSOSE.2007.4304229.

Budi, S., P. De Souza, G. Timms, V. Malhotra, and P. Turner (Nov. 2014). "Optimisation in the Design of Environmental Sensor Networks with Robustness Consideration". In: *Sensors* 15. DOI: 10.3390/s151229765.

Chen, D., C. Lee, C. Park, and P. Mendes (Oct. 2007). "Parallelizing simulated annealing algorithms based on high-performance computer". In: *Journal of Global Optimization* 39.2, pp. 261–289. ISSN: 1573-2916. DOI: 10.1007/s10898-007-9138-0. URL: https://doi.org/10.1007/s10898-007-9138-0.

Chen, S., I. Gorton, A. Liu, and Y. Basser (Jan. 2002). "Performance prediction of COTS component-based enterprise applications". In: *ICSE 2002 Workshop: CBSE5:1-7*.

Chen, S., Y. Liu, I. Gorton, and A. Liu (2005). "Performance prediction of component-based applications". In: *Journal of Systems and Software* 74.1. Automated Component-Based Software Engineering, pp. 35 –43. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2003.05.005. URL: http://www.sciencedirect.com/science/article/pii/S0164121203003200.

Chen, W., X. Qiao, J. Wei, and T. Huang (2012). "A Profit-Aware Virtual Machine Deployment Optimization Framework for Cloud Platform Providers". In: *2012 IEEE Fifth International Conference on Cloud Computing*, pp. 17–24.

Chise, C. and I. Jurca (May 2009). "Towards early performance assessment based on UML MARTE models for distributed systems". In: *2009 5th International Symposium on Applied Computational Intelligence and Informatics*, pp. 521–526. DOI: 10.1109/SACI.2009.5136304.

Ciancone, A., M.L. Drago, A. Filieri, V. Grassi, H. Koziolek, and R. Mirandola (2014). "The KlaperSuite framework for model-driven reliability analysis of component-based systems". In: *Software Systems Modeling*. ISSN: 1619-1374. DOI: 10.1007/s10270-013-0334-8. URL: https://doi.org/10.1007/s10270-013-0334-8.

Ciancone, A., A. Filieri, M. Drago, R. Mirandola, and V. Grassi (2011). "KlaperSuite: An Integrated Model-Driven Environment for Reliability and Perfor-

mance Analysis of Component-Based Systems". In: *Objects, Models, Components, Patterns*. Ed. by J. Bishop and A. Vallecillo. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 99–114. ISBN: 978-3-642-21952-8.

Cook, S.C. (2001). "2.3.1 On the Acquisition of Systems of Systems". In: *INCOSE International Symposium* 11.1, pp. 383–390. ISSN: 2334-5837. DOI: 10.1002/j.2334-5837.2001.tb02318.x. URL: http://dx.doi.org/10.1002/j.2334-5837.2001.tb02318.x.

Cortellessa, V. and R. Mirandola (2002). "PRIMA-UML: a performance validation incremental methodology on early UML diagrams". In: *Science of Computer Programming* 44.1. Special Issue on Unified Modeling Language (UML 2000), pp. 101 –129. ISSN: 0167-6423. DOI: https://doi.org/10.1016/S0167-6423(02)00033-3. URL: http://www.sciencedirect.com/science/article/pii/S0167642302000333.

Curnow, H.J. and B.A. Wichmann (1976). "A Synthetic Benchmark". In: *Compute. J.* 19, pp. 43–49.

D'Ambros, D., M. Lanza, and R. Robbes (2010). "An extensive comparison of bug prediction approaches". In: *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pp. 31–41. DOI: 10.1109/MSR.2010.5463279.

Davis, J. (2003). "GME: The Generic Modeling Environment". In: *Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*. OOPSLA 03. Anaheim, CA, USA: ACM, pp. 82–83. ISBN: 1-58113-751-6. DOI: 10.1145/949344.949360. URL: http://doi.acm.org/10.1145/949344.949360.

De Jong, K. (Jan. 2006). *Evolutionary Computation – A Unified Approach*. ISBN: 978-0-262-04194-2.

Dekking, F.M., C. Kraaikamp, H.P. Lopuhaä, and L.E. Meester (2006). *A Modern Introduction to Probability and Statistics: Understanding Why and How*. Springer Texts in Statistics. Springer London. ISBN: 9781846281686. URL: https://books.google.com.au/books?id=TEcmHJX67coC.

Denaro, G., A. Polini, and W. Emmerich (2004). "Early performance testing of distributed software applications". In: *SIGSOFT Software Engineering Notes* 29, pp. 94–103.

Ding, J. (2016). "An approach for modeling and analyzing dynamic software architectures". In: *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, pp. 2086–2092.

Distefano, S., D. Paci, A. Puliafito, and M. Scarpa (2004). "UML Design and Software Performance Modeling". In: *Computer and Information Sciences -*

ISCIS 2004. Ed. by Cevdet Aykanat, Tugrul Dayar, and Ibrahim Korpeoglu. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 564–573. ISBN: 978-3-540-30182-0.

Do, Q., S. Cook, and M. Lay (2014). "An Investigation of MBSE Practices across the Contractual Boundary". In: *Procedia Computer Science* 28. 2014 Conference on Systems Engineering Research, pp. 692–701. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2014.03.083. URL: https://www.sciencedirect.com/science/article/pii/S187705091400146X.

Edwards, G., S. Malek, and N. Medvidovic (2007a). "Scenario-Driven Dynamic Analysis of Distributed Architectures". In: *Proc. of the 10th Intl Conf. on Fundamental Approaches to Software Engineering*, pp. 125–139.

Edwards, G., S. Malek, and N. Medvidovic (2007b). "Scenario-Driven Dynamic Analysis of Distributed Architectures". In: *Fundamental Approaches to Software Engineering*. Ed. by Antonia Dwyer Matthew B.and Lopes. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 125–139. ISBN: 978-3-540-71289-3.

Eiben, A. and J. Smith (Jan. 2003). *Introduction To Evolutionary Computing*. Vol. 45. ISBN: 978-3-642-07285-7. DOI: 10.1007/978-3-662-05094-1.

Eismann, S., J. Walter, J. von Kistowski, and S. Kounev (2018). "Modeling of Parametric Dependencies for Performance Prediction of Component-Based Software Systems at Run-Time". In: *2018 IEEE International Conference on Software Architecture (ICSA)*, pp. 135–13509. DOI: 10.1109/ICSA.2018.00023.

Evangelista, C. (2021). "Performance modelling of NoSQL DBMS". In: *Proceedings of the 2020 OMI Seminars (PROMIS 2020)*. Vol. 1. Universität Ulm, pp. 6–1.

Faisal, Adnan, Dorina Petriu, and Murray Woodside (2014). "A systematic approach for composing general middleware completions to performance models". In: *European Workshop on Performance Engineering*. Springer, pp. 30–44.

Falkner, K., V. Chiprianov, N. Falkner, C. Szabo, J. Hill, G. Puddy, D. Fraser, A. Johnston, M. Rieckmann, and A. Wallis (July 2013). "Model-driven performance prediction of distributed real-time embedded defence systems". In: *In Proceedings of the 18th International Conference on Engineering of Complex Computer Systems (ICECCS 2013)*.

Falkner, K., V. Chiprianov, N. Falkner, C. Szabo, and G. Puddy (Jan. 2014). "A model-driven engineering method for DRE defense systems performance analysis and prediction". In: pp. 301–326. DOI: 10.4018/978-1-4666-6194-3.ch012.

Falkner, K., C. Szabo, V. Chiprianov, G. Puddy, M. Rieckmann, D. Fraser, and C. Aston (May 2018). "Model-driven performance prediction of systems of systems". In: *Software Systems Modeling* 17.2, pp. 415–441. ISSN: 1619-1374. DOI: 10.1007/s10270-016-0547-8. URL: https://doi.org/10.1007/s10270-016-0547-8.

Farcas, C., E. Farcas, I Krueger, and M. Menarini (Apr. 2010). "Addressing the integration challenge for avionics and automotive systems: From components to rick services". In: *Proceedings of the IEEE* 98.4, pp. 562–583.

Feiler, P.H., D.P. Gluch, and J.J. Hudak (2006). "The Architecture Analysis Design Language (AADL): An Introduction". In:

Felemban, E., F.K. Shaikh, U.M. Qureshi, A.A. Sheikh, and S.B. Qaisar (2015). "Underwater Sensor Network Applications: A Comprehensive Survey". In: *International Journal of Distributed Sensor Networks* 11.11, p. 896832. DOI: 10.1155/2015/896832. eprint: https://doi.org/10.1155/2015/896832. URL: https://doi.org/10.1155/2015/896832.

Franco, J., R. Barbosa, and M. Zenha-Rela (Apr. 2013). "Reliability Analysis of Software Architecture Evolution". In: DOI: 10.1109/LADC.2013.16.

Friedenthal, S., R. Griego, and M. Sampson (Jan. 2009). "INCOSE Model Based Systems Engineering (MBSE) Initiative". In:

Gilmore, S., L. Gonczy, N. Koch, P. Mayer, M. Tribastone, and D. Varro (2011). "Non-functional properties in the model-driven development of service-oriented systems". In: *Software Systems Modeling* 10.3, pp. 287–311. ISSN: 1619-1374. DOI: 10.1007/s10270-010-0155-y. URL: https://doi.org/10.1007/s10270-010-0155-y.

Godfrey, M. and Q. Tu (2001). "Growth, Evolution, and Structural Change in Open Source Software". In: *Proceedings of the 4th International Workshop on Principles of Software Evolution*. IWPSE 01. Vienna, Austria: ACM, pp. 103–106. ISBN: 1-58113-508-4. DOI: 10.1145/602461.602482. URL: http://doi.acm.org/10.1145/602461.602482.

Gokhale, A., D. Schmidt, T. Lu, B. Natarajan, and N. Wang (Jan. 2003). "CoSMIC: An MDA Generative Tool for Distributed Real-time and Embedded Applications." In: pp. 300–306.

Grassi, V. and R. Mirandola (Jan. 2002). "PRIMAmob-UML: a methodology for performance analysis of mobile software architectures." In: pp. 262–274. DOI: 10.1145\/584369.584411.

Grassi, V., R. Mirandola, E. Randazzo, and A. Sabetta (2008). "KLAPER: An Intermediate Language for Model-Driven Predictive Analysis of Performance and Reliability". In: *The Common Component Modeling Example: Comparing*

Software Component Models. Ed. by Andreas Rausch, Ralf Reussner, Raffaela Mirandola, and Frantivsek Plavsil. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 327–356. ISBN: 978-3-540-85289-6. DOI: 10.1007/978-3-540-85289-6_13. URL: https://doi.org/10.1007/978-3-540-85289-6_13.

Grassi, V., R. Mirandola, and A. Sabetta (2005). "From Design to Analysis Models: A Kernel Language for Performance and Reliability Analysis of Component-based Systems". In: *Proceedings of the 5th International Workshop on Software and Performance*. WOSP 05. Palma, Illes Balears, Spain: ACM, pp. 25–36. ISBN: 1-59593-087-6. DOI: 10.1145/1071021.1071024. URL: http://doi.acm.org/10.1145/1071021.1071024.

Grassi, V., R. Mirandola, and A. Sabetta (Apr. 2007). "Filling the Gap Between Design and PerformanceReliability Models of Component-based Systems". In: *J. Syst. Softw.* 80.4, pp. 528–558. ISSN: 0164-1212. DOI: 10.1016/j.jss.2006.07.023. URL: http://dx.doi.org/10.1016/j.jss.2006.07.023.

Happe, J., H. Koziolek, and R. Reussner (2011). "Facilitating Performance Predictions Using Software Components". In: *IEEE Software* 28, pp. 27–33.

Heidemann, J., W. Ye, J. Wills, A. Syed, and Y. Li (2006). "Research challenges and applications for underwater sensor networking". In: *IEEE Wireless Communications and Networking Conference, 2006. WCNC 2006.* Vol. 1, pp. 228–235.

Heinrich, R., D. Werle, H. Klare, R. Reussner, M. Kramer, S. Becker, J. Happe, H. Koziolek, and K. Krogmann (2018). "The palladio-bench for modeling and simulating software architectures". In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pp. 37–40.

Hemer, D. and Y. Ding (2009). "Modelling Software Architectures Using CRADLE". In:

Henderson, K. and A. Salado (2021). "Value and benefits of model-based systems engineering (MBSE): Evidence from the literature". In: *Systems Engineering* 24.1, pp. 51–66. DOI: https://doi.org/10.1002/sys.21566. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sys.21566. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sys.21566.

Hennig, C., T. Hoppe, H. Eisenmann, A. Viehl, and O. Bringmann (Feb. 2016). "SCDML: A language for Conceptual Data Modeling in Model-based Systems Engineering". In: *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pp. 184–192.

Herrera, F., H. Posadas, P. Peñil, E. Villar, F. Ferrero, R. Valencia, and G. Palermo (2014). "The COMPLEX methodology for UML/MARTE Modeling and design

space exploration of embedded systems". In: *Journal of Systems Architecture* 60.1, pp. 55–78. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2013.10.003. URL: https://www.sciencedirect.com/science/article/pii/S138376211300194X.

Heydari, B., M. Mosleh, and K. Dalili (2016). "From Modular to Distributed Open Architectures: A Unified Decision Framework". In: *Systems Engineering* 19.3, pp. 252–266. DOI: https://doi.org/10.1002/sys.21348. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sys.21348. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sys.21348.

Hill, J. and A. Gokhale (Jan. 2008). "Towards Improving End-to-End Performance of Distributed Real-Time and Embedded Systems Using Baseline Profiles". In: vol. 150, pp. 43–57. DOI: 10.1007/978-3-540-70561-1_4.

Hill, J.H., D.C Schmidt, J. Edmondson, and A. Gokhale (2010). "Tools for Continuously Evaluating Distributed System Qualities". In: *IEEE Software* 27.4, pp. 65–71.

Hill, J.H., D.C. Schmidt, and J.M. Slaby (2009). "Evaluating Quality of Service for Enterprise Distributed System". In: 37, pp. 335–371. DOI: 10.4018/978-1-59904-699-0.ch011.

Holland, J.H. (Dec. 1975). *Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Ann Arbor, MI, United States: The University of Michigan Press. ISBN: 0472084607.

Hoos, H. and T. Sttzle (2004). *Stochastic Local Search: Foundations amp; Applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 1558608729.

Hugues, J., B. Zalila, L. Pautet, and F. Kordon (Jan. 2007). "Rapid Prototyping of Distributed Real-Time Embedded Systems Using the AADL and Ocarina." In: pp. 106–112.

Iakushkin, Oleg, Yulia Shichkina, and Olga Sedova (2016). "Petri nets for modelling of message passing middleware in cloud computing environments". In: *International Conference on Computational Science and Its Applications*. Springer, pp. 390–402.

Jamal, A. (Aug. 2019). "Global Optimization Using Local Search Approach for Course Scheduling Problem". In: DOI: 10.5772/intechopen.86228.

Jasmine, K.S. and R. Vasantha (2007). "Design Based Performance Prediction of Component Based Software Products". In: *World Academy of Science, Engineering and Technology* 30, pp. 266–269.

Jindal, H., S. Saxena, and S. Singh (2014). "Challenges and issues in underwater acoustics sensor networks: A review". In: *2014 International Conference on Parallel, Distributed and Grid Computing*, pp. 251–255.

Kalske, M., N. Mäkitalo, and T. Mikkonen (2018). "Challenges When Moving from Monolith to Microservice Architecture". In: *Current Trends in Web Engineering*. Ed. by I. Garrigós and M. Wimmer. Cham: Springer International Publishing, pp. 32–47. ISBN: 978-3-319-74433-9.

Kaur, A. (2012). "Application Of UML In Real-Time Embedded Systems". In: *International Journal of Software Engineering Applications (IJSEA)*. URL: https://ssrn.com/abstract=3816281.

Kent, Stuart (2002). "Model Driven Engineering". In: *Integrated Formal Methods*. Ed. by M. Butler, L. Petre, and K. Sere. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 286–298. ISBN: 978-3-540-47884-3.

Kerr, C.I.V., R. Phaal, and D.R. Probert (2008). "Technology insertion in the defence industry: A primer". In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 222.8, pp. 1009–1023. DOI: 10.1243/09544054JEM1080. eprint: https://doi.org/10.1243/09544054JEM1080. URL: https://doi.org/10.1243/09544054JEM1080.

Kewley, R., J. Cook, N. Goerger, D. Henderson, and E. Teague (2008). "Federated Simulations for Systems of Systems Integration". In: *Proceedings of the 2008 Winter Simulation Conference*, pp. 1121–1129.

Koziolek, H. (2010). "Performance evaluation of component-based software systems: A survey". In: *Performance Evaluation* 67.8. Special Issue on Software and Performance, pp. 634 –658. ISSN: 0166-5316. DOI: https://doi.org/10.1016/j.peva.2009.07.007. URL: http://www.sciencedirect.com/science/article/pii/S016653160900100X.

Koziolek, H., B. Schlich, S. Becker, and M. Hauck (2013). "Performance and reliability prediction for evolving service-oriented software systems". In: *Empirical Software Engineering* 18.4, pp. 746–790.

Koziolek, H., B. Schlich, C. Bilich, R. Weiss, S. Becker, K. Krogmann, M. Trifu, R. Mirandola, and A. Koziolek (May 2011). "An industrial case study on quality impact prediction for evolving service-oriented software". In: *2011 33rd International Conference on Software Engineering (ICSE)*, pp. 776–785.

Kuperberg, M., K. Krogmann, and R. Reussner (2008). "Performance Prediction for Black-Box Components Using Reengineered Parametric Behaviour Models". In: *Proceedings of the 11th International Symposium on Component-Based Software Engineering*. CBSE '08. Karlsruhe, Germany: Springer-Verlag, pp. 48–63.

ISBN: 978-3-540-87890-2. DOI: 10.1007/978-3-540-87891-9_4. URL: http://dx.doi.org/10.1007/978-3-540-87891-9_4.

Kwiatkowska, M., G. Norman, and D. Parker (2009). "PRISM: Probabilistic Model Checking for Performance and Reliability Analysis". In: *ACM SIGMETRICS Performance Evaluation Review* 36.4, pp. 40–45.

Lai, X, A. Balakrishnan, T. Lange, M. Jenihhin, T. Ghasempouri, J. Raik, and D. Alexandrescu (2019). "Understanding multidimensional verification: Where functional meets non-functional". In: *Microprocessors and Microsystems* 71, p. 102867. ISSN: 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2019.102867. URL: https://www.sciencedirect.com/science/article/pii/S0141933119300250.

Lavenberg, S.S. (1983). *Computer Performance Modeling Handbook*. Orlando, FL, USA: Academic Press, Inc. ISBN: 0124387209.

Lavery, J., C. Boldyreff, B. Ling, and C. Allison (2004). "Modelling the evolution of legacy systems to Web-based systems". In: *Journal of Software Maintenance and Evolution: Research and Practice* 16.1-2, pp. 5–30. DOI: https://doi.org/10.1002/smr.282. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.282. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.282.

Li, C., W. Wang, and Y. Shi (2013). "Performance Prediction Method for UML Software Architecture and Its Automation [J]". In: *Journal of Software* 7.

Li, J., J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai (2009). "Performance model driven QoS guarantees and optimization in clouds". In: *2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pp. 15–22.

Li, Z., L. Tan, X. Wang, S. Lu, Y. Zhou, and C. Zhai (2006). "Have Things Changed Now?: An Empirical Study of Bug Characteristics in Modern Open Source Software". In: *Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability*. ASID '06. San Jose, California: ACM, pp. 25–33. ISBN: 1-59593-576-2. DOI: 10.1145/1181309.1181314. URL: http://doi.acm.org/10.1145/1181309.1181314.

Liehr, A. and K. Buchenrieder (Feb. 2009). "Transforming UML-Based System Descriptions into Simulation Models as Part of System Development Frameworks". In: vol. 5717, pp. 857–864. DOI: 10.1007/978-3-642-04772-5_110.

Little, J. and S. Graves (July 2008). "Little's Law". In: pp. 81–100. DOI: 10.1007/978-0-387-73699-0_5.

Liu, T., T. Lu, and Z. Liu (2010). "An optimization approach for service deployment in service oriented clouds". In: *The 2010 14th International Conference on Computer Supported Cooperative Work in Design*, pp. 390–395.

Liu, Y., A. Fekete, and I. Gorton (Jan. 2004). "Predicting the Performance of Middleware-based Applications at the Design Level". In: *SIGSOFT Softw. Eng. Notes* 29.1, pp. 166–170. ISSN: 0163-5948. DOI: 10.1145/974043.974071. URL: http://doi.acm.org/10.1145/974043.974071.

Madni, A. and M. Sievers (Sept. 2014). "System of Systems Integration: Key Considerations and Challenges". In: *Systems Engineering* 17. DOI: 10.1002/sys.21272.

Maier, M.W. (1998). "Architecting principles for systems-of-systems". In: *Systems Engineering* 1.4, pp. 267–284. ISSN: 1520-6858. DOI: 10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D.

Manthorpe, W.H.J. (1996). "The emerging joint system of systems: A systems engineering challenge and opportunity for APL". In: *John Hopkins APL Tech Dig*, pp. 305–310.

Martens, N. (2011). "The impact of non-functional requirements on project success". In: *Utrecht University, Msc Thesis, Utrecht*.

Medvidovic, N., D.S. Rosenblum, and R.N. Taylor (1999). "A Language and Environment for Architecture-based Software Development and Evolution". In: *Proceedings of the 21st International Conference on Software Engineering*. ICSE 99. Los Angeles, California, USA: ACM, pp. 44–53. ISBN: 1-58113-074-0. DOI: 10.1145/302405.302410. URL: http://doi.acm.org/10.1145/302405.302410.

Meedeniya, I., B. Buhnova, A. Aleti, and G. Lars (2011). "Reliability-driven deployment optimization for embedded systems". In: *Journal of Systems and Software* 84.5, pp. 835 –846. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2011.01.004. URL: http://www.sciencedirect.com/science/article/pii/S0164121211000069.

Merola, L. (May 2006). "The COTS software obsolescence threat". In: *Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems*.

Mitchell, S. (Sept. 2010a). "Complex Product Family Modeling for Common Submarine Combat System MBSE." In: *Third International Conference on Model Based Systems Engineering*.

Mitchell, S. (May 2010b). "Model-Based System Development for Managing the Evolution of a Common Submarine Combat System". In:

Mitchell, S. (May 2011). "Efficient Management of Configurations in the Model-Based System Development of a Common Submarine Combat System". In: *AFCEA-GMU Critical Issues in C4I Symposium*.

Mohagheghi, P., W. Gilani, A. Stefanescu, M.A. Fernandez, B. Nordmoen, and M. Fritzsche (2013). "Where does model-driven engineering help? Experiences from three industrial cases". In: *Software Systems Modeling*. ISSN: 1619-1374. DOI: 10.1007/s10270-011-0219-7. URL: https://doi.org/10.1007/s10270-011-0219-7.

"Book Alert: The Art of Computer Systems Performance Analysis: Techniques for Experirnental Design, Measurement, Simulation and Modeling, by Raj Jain" (Dec. 1990). In: *SIGSIM Simul. Dig.* 21.2. Ed. by R. E. Nance, pp. 53–. ISSN: 0163-6103. DOI: 10.1145/382264.1108821. URL: http://doi.acm.org/10.1145/382264.1108821.

NASA (2005). *Exploration System of Systems Programmatic Requirements and Guidelines Document –Revision E*. Tech. rep. ESMD-RQ-0021. National Aeronautics and Space Administration.

Nawinna, Dasuni and Shashi Sandeepani (Dec. 2020). "Impact of Non-Functional Requirements on the Success of Ubiquitous Systems". In: DOI: 10.1109/ICAC51239.2020.9357287.

Nazari, A., D. Thiruvady, A. Aleti, and I. Moser (2016). "A mixed integer linear programming model for reliability optimisation in the component deployment problem". In: *Journal of the Operational Research Society* 67.8, pp. 1050–1060.

NSWCDD (2004). *Open Architecture (OA) Computing Environment Design Guidance*. URL: https://cno.ahf.nmci.navy.mil/n2n6/a_webdoc01.nsf/0D2C512E68790C70852571F400760404/File/OACEDesignGuidance.pdf (visited on 02/18/2013).

OMG (2003). *UML Profile for Schedulability, Performance, and Time, v1.0*. OMG.

OMG (2011). *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems V 1.1*. OMG.

Ovaliadis, K., N. Savage, and V. Kanakaris (2010). "Energy efficiency in underwater sensor networks: a research review". English. In: *Journal of Engineering Science and Technology Review (JESTR)* 3.1, pp. 151–156. ISSN: 1791-2377.

Ozkaya, M. (2018). "ANALYSING UML-BASED SOFTWARE MODELLING LANGUAGES". In: *Journal of Aeronautics and Space Technologies* 11.2, pp. 119–134. URL: http://193.255.215.140/index.php/JAST/article/view/326.

Paunov, S., J. Hill, D.C. Schmidt, S. Baker, and J. Slaby (2006). "Domain-Specific Modeling Languages for Configuring and Evaluating Enterprise DRE System Quality of Service". In: *13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*.

Peng, W., H. Li, M. Yao, and Z. Sun (2010). "Deployment optimization for AUTOSAR system configuration". In: *2010 2nd International Conference on Computer Engineering and Technology*. Vol. 4, pp. V4–189–V4–193.

Perez, J., J. Diaz, J. Garbajosa, A. Yague, E. Gonzalez, and M. Lopez-Perea (2013). "Large-scale Smart Grids As System of Systems". In: *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems*. SESoS 13. Montpellier, France: ACM, pp. 38–42. ISBN: 978-1-4503-2048-1. DOI: 10.1145/2489850.2489858. URL: http://doi.acm.org/10.1145/2489850.2489858.

Perez, R.M. (2014). "Application of MBSE to Risk-Informed Design Methods for Space Mission Applications". In: DOI: 10.2514/6.2014-4411. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2014-4411.

Petersen, B., T. Petersen, P. Andersen, M. Nielsen, and C. Lundegaard (2009). "A generic method for assignment of reliability scores applied to solvent accessibility predictions". In: *BMC Structural Biology*. DOI: 10.1186/1472-6807-9-51.

Pratt, G.C., C.Y. Wu, D. Bock, J.L. Adgate, G. Ramachandran, T.H. Stock, M. Morandi, and K. Sexton (2004). "Comparing Air Dispersion Model Predictions with Measured Concentrations of VOCs in Urban Communities". In: *Environmental Science Technology* 38.7, pp. 1949–1959. DOI: 10.1021/es0306381. URL: https://doi.org/10.1021/es0306381.

Rathfelder, C., B. Klatt, K. Sachs, and S. Kounev (2014). "Modeling event-based communication in component-based software architectures for performance predictions". In: *Software Systems Modeling*. ISSN: 1619-1374.

Rebai, M., M. Le berre, H. Snoussi, F. Hnaien, and L. Khoukhi (2015). "Sensor deployment optimization methods to achieve both coverage and connectivity in wireless sensor networks". In: *Computers Operations Research* 59. ISSN: 0305-0548. URL: http://www.sciencedirect.com/science/article/pii/S0305054814002780.

Reussner, Ralf, Steffen Becker, Erik Burger, Jens Happe, Michael Hauck, Anne Koziolek, Heiko Koziolek, Klaus Krogmann, and Michael Kuperberg (2011). "The Palladio component model". In:

SAE (2007). *Architecture Analysis Design Language (AS5506)*. SAE.

Sage, A.P. and C.D. Cuppan (2001). "On the Systems engineering and management of systems of systems and federation of systems". In: *Information, Knowledge and Systems Management* 2, pp. 325–345.

Santtila, M., K. Häkkinen, K. Pihlainen, and H. Kyröläinen (Feb. 2013). "Comparison Between Direct and Predicted Maximal Oxygen Uptake Measurement During Cycling". In: *Military Medicine* 178.2, pp. 234–238. ISSN: 0026-4075. DOI: 10.7205/MILMED-D-12-00276. URL: https://doi.org/10.7205/MILMED-D-12-00276.

Saunders, S. (2003). "Reducing Project Re-Work By The Use Of Model Based Systems Engineering Processes and Tools". In: *Proceedings of the Joint Systems Engineering Test and Evaluation Conference SETE*.

Selic, B. (1998). "Using UML for modeling complex real-time systems". In: *Languages, Compilers, and Tools for Embedded Systems*. Ed. by Frank Mueller and Azer Bestavros. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 250–260. ISBN: 978-3-540-49673-1.

Shailesh, T., A. Nayak, and D. Prasad (2020). "An UML Based Performance Evaluation of Real-Time Systems Using Timed Petri Net". In: *Computers* 9.4. ISSN: 2073-431X. DOI: 10.3390/computers9040094. URL: https://www.mdpi.com/2073-431X/9/4/94.

Slaby, J., S. Baker, J. Hill, and D. Schmidt (Jan. 2006). "Applying System Execution Modeling Tools to Evaluate Enterprise Distributed Real-time and Embedded System QoS". In: *Proceedings of the 12th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 350–362. DOI: 10.1109/RTCSA.2006.17.

Smith, C.U. (1990). *Performance Engineering of Software Systems*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201537699.

Smith, C.U. and L.G. Williams (1997). "Performance engineering evaluation of object-oriented systems with SPE EDTM". In: *Computer Performance Evaluation Modelling Techniques and Tools*. Ed. by Raymond Marie, Brigitte Plateau, Maria Calzarossa, and Gerardo Rubino. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 135–154.

Sutton, A.M., F. Neumann, and S. Nallaperuma (2014). "Parameterized Runtime Analyses of Evolutionary Algorithms for the Planar Euclidean Traveling Salesperson Problem". In: *Evolutionary Computation* 22.4, pp. 595–628. DOI: 10.1162/EVCO_a_00119.

Tabuchi, N., N. Sato, and H. Nakamura (2005). "Model-Driven Performance Analysis of UML Design Models Based on Stochastic Process Algebra". In: *Model*

Driven Architecture – Foundations and Applications. Ed. by Alan Hartman and David Kreische. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 41–58.

Ulbrich, Peter, Christoph Elsner, Martin Hoffmann, R. Schmid, and W. Schröder-Preikschat (2010). "Using MARTE in code-centric real-time projects providing evolution support". In: *Proceedings of the 1st Workshop on Model Based Engineering for Embedded Systems Design (M-BED'10)*, pp. 25–29.

Vanderbilt (2008). *Generic Modeling Environment (GME)*. Vanderbilt. URL: https://www.isis.vanderbilt.edu/Projects/gme/.

Vanrompay, Y., P. Rigole, and Y. Berbers (2008). "Genetic Algorithm-Based Optimization of Service Composition and Deployment". In: *Proceedings of the 3rd International Workshop on Services Integration in Pervasive Environments*. SIPE '08. Sorrento, Italy: Association for Computing Machinery, 13–18. ISBN: 9781605582085. DOI: 10.1145/1387309.1387313. URL: https://doi.org/10.1145/1387309.1387313.

Vikhar, P.A. (2016). "Evolutionary algorithms: A critical review and its future prospects". In: *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pp. 261–265. DOI: 10.1109/ICGTSPICC.2016.7955308.

Wang, J., X. Jin, P. Zeng, M. Wan, and C. Xia (2017). "Deployment optimization for a long-distance wireless backhaul network in industrial cyber physical systems". In: *International Journal of Distributed Sensor Networks* 13.11, p. 1550147717744993. DOI: 10.1177/1550147717744993. eprint: https://doi.org/10.1177/1550147717744993. URL: https://doi.org/10.1177/1550147717744993.

Weiderman, N.H., J.K. Bergey, D.B. Smith, and S.R. Tilley (1997). *Approaches to Legacy System Evolution*. Tech. rep. CMU/SEI-97-TR-014. Software Engineering Institute, Carnegie-Mellon University.

Weise, T., Y. Wu, R. Chiong, K. Tang, and J. Lassig (Nov. 2016). "Global versus local search: the impact of population sizes on evolutionary algorithm performance". In: *Journal of Global Optimization* 66.3, pp. 511–534. ISSN: 1573-2916. DOI: 10.1007/s10898-016-0417-5. URL: https://doi.org/10.1007/s10898-016-0417-5.

White, J., B. Dougherty, C. Thompson, and D.C. Schmidt (Sept. 2011). "ScatterD: Spatial Deployment Optimization with Hybrid Heuristic Evolutionary Algorithms". In: *ACM Trans. Auton. Adapt. Syst.* 6.3. ISSN: 1556-4665. DOI: 10.1145/2019583.2019585. URL: https://doi.org/10.1145/2019583.2019585.

Wong, H., H. Marie-Nelly, S. Herbert, P. Carrivain, H. Blanc, R. Koszul, E. Fabre, and C. Zimmer (2012). "A Predictive Computational Model of the Dynamic 3D Interphase Yeast Nucleus". In: *Current Biology* 22.20, pp. 1881 –1890. ISSN: 0960-9822. DOI: https://doi.org/10.1016/j.cub.2012.07.069. URL: http://www.sciencedirect.com/science/article/pii/S096098221200927X.

Zhang, Y., N. Chen, J. Wei, and T. Huang (2006). "Completing UML Model of Component-Based System with Middleware for Performance Evaluation". In: *Emerging Directions in Embedded and Ubiquitous Computing*. Ed. by Xiaobo Zhou, Oleg Sokolsky, Lu Yan, Eun-Sun Jung, Zili Shao, Yi Mu, Dong Chun Lee, Dae Young Kim, Young-Sik Jeong, and Cheng-Zhong Xu. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 72–82. ISBN: 978-3-540-36851-9.

Zhao, C., J. Kong, J. Dong, and K. Zhang (2007). "Pattern-based design evolution using graph transformation". In: *Journal of Visual Languages Computing* 18.4. Visual Interactions in Software Artifacts, pp. 378–398. ISSN: 1045-926X. DOI: https://doi.org/10.1016/j.jvlc.2007.07.004. URL: https://www.sciencedirect.com/science/article/pii/S1045926X07000407.

Zhu, J., Z. Zheng, Y. Zhou, and M.R. Lyu (2013). "Scaling Service-Oriented Applications into Geo-distributed Clouds". In: *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, pp. 335–340.

Zimmer, B., S. Burklen, J. Hofflinger, M. Trapp, and P. Liggesmeyer (2012). "Safety-Focused Deployment Optimization in Open Integrated Architectures". In: *Computer Safety, Reliability, and Security*. Ed. by Frank Ortmeier and Peter Daniel. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 328–339. ISBN: 978-3-642-33678-2.

A. Modelling Technical Detail

A.1 Existing MEDEA modelling element predefined attributes

New Attribute	Description	Associating Model Element
maxLatencyStringGroupDetail	This data pair is id, time in milliseconds i.e., 1,650	Assemblies Elements
assemblyMandateDeployGroup	This is just an unique id label	Assembly Elements
noColocationPairId	This is just a unique id label	Component Elements
componentMandateDeployGroup	This is just a unique id label	Component Elements
componentMandateDeployOnly	This is just a unique id label	Component Elements
connectionFrequency	This can be a number or a function using the variable 'f'. NB: the above detail are placed on the delegate to delegate connection when there are delegates nothing is added to the eventPort to delegate connectionFrequency	Connection Elements
connectionPayload	This can be a number or a function using the variable 'n' or 'm'. NB: the above detail are placed on the delegate to delegate connection when there are delegates nothing is added to the eventPort to delegate connectionFrequency	Connection Elements
internalConnection	This is boolean flag and set with the string 'true' or 'false'. NB: the above detail are placed on the delegate to delegate connection when there are delegates nothing is added to the eventPort to delegate connectionFrequency	CriticalChain Connection Element

New Attribute	Description	Associating Model Element
maxLatencyConnection	This is a number representing the required time in milliseconds required for particular connection's latency. NB: the above detail are placed on the delegate to delegate connection when there are delegates nothing is added to the eventPort to delegate connection-Frequency	CriticalChain Connection Element
maxLatencyStringGroup	This is an unique id label to be applied to the connections part of a particular string of interest which has its latency set at the assembly element level. NB: the above detail are placed on the delegate to delegate connection when there are delegates nothing is added to the eventPort to delegate connection-Frequency	CriticalChain Connection Element
spareCapacityCPU	this is a number representing the percentage size of spare CPU resources to remain on the particular node	Node elements
spareCapacityMemory	This is a number representing the percentage size of spare Memory resources to remain on the particular node	Node elements

New Attribute	Description	Associating Model Element
spareCapacityCPUWeight	this a number representing the weight to be applied to spare CPU capacity result	Node elements
spareCapacityMemoryWeight	This a number representing the weight to be applied to spare Memory capacity result	Node elements
assemblyMandateDeployGroup	This is an unique label that lines up with the particular labels applied to assembly elements	Node elements
componentMandateDeployGroup	This is an unique label that lines up with the particular labels applied to the component elements	Node elements
componentMandateDeployOnly	This is an unique label that lines up with the particular labels applied to the component elements	Node elements
connectionSpeedMB	This is number representing the speed of the particular connection. NB: this may be taken over by the ZMQ testing framework and result provide for bandwidth and through put for ZMQ connections	Node element

A.2 Text-based Modelling Elements

CLI Prompt	Attribute Application	Non-Function Characteristics	Pictorial-model entity complement
'Change the double bit size'	Used to set the number of bits that will represent a double primitive within a message	Size of message primitive	Used in conjunction with the message payload attribute for the interface connection model entity within the Assembly context window. NB: Check if this is still used or has been deprecated
'Change the integer bit size'	Used to set the number of bits that will represent an integer primitive within a message	Size of message primitive	Used in conjunction with the message payload attribute for the interface connection model entity within the Assembly context window. NB: Check if this is still used or has been deprecated.
'Change the flag to indicate whether all the connection latency requirements are needed'	A Boolean flag used to tell the optimisation process to either ignore the deployment if profiles that do not satisfy the connection latency requirements if set to TRUE or use the current deployment profiles do not satisfy the connection latency requirements if set to FALSE	Decision flag to determine if certain deployment options will be part of the optimisation processing or not	NA

CLI Prompt	Attribute Application	Non-Function Characteristics	Pictorial-model entity complement
'Change the flag to indicate whether all the component mandate deployment requirements are needed'	A Boolean flag used to tell the optimisation process to either ignore the deployment if profiles that do not satisfy the component mandate deployment requirements if set to TRUE or use the current deployment profiles do not satisfy the connection latency requirements if set to FALSE	Decision flag to determine if certain deployment options will be part of the optimisation processing or not	NA
'Change the flag to indicate whether all the assembly mandate deployment requirements are needed'	A Boolean flag used to tell the optimisation process to either ignore the deployment if profiles that do not satisfy the assembly mandate deployment requirements if set to TRUE or use the current deployment profiles do not satisfy the connection latency requirements if set to FALSE	Decision flag to determine if certain deployment options will be part of the optimisation processing or not	NA

CLI Prompt	Attribute Application	Non-Function Characteristics	Pictorial-model entity complement
'Change the flag to indicate whether all the no colocation requirements are needed'	A Boolean flag used to tell the optimisation process to either ignore the deployment if profiles that do not satisfy the component no colocation mandate deployment requirements if set to TRUE or use the current deployment profiles do not satisfy the connection latency requirements if set to FALSE	Decision flag to determine if certain deployment options will be part of the optimisation processing or not	NA
'Change the weight setting for thread execution'	A multiplication factor used to define the importance of thread execution latency optimisation score and overall influence of the score for the overall optimisation score and therefore the population search direction	Importance of thread execution	NA
'Change the satisfied weight setting for connection latency'	A multiplication factor used to define the importance of the connection latency optimisation score and overall influence of the score for the overall optimisation score and therefore the population search direction	Importance of connection latency	NA

CLI Prompt	Attribute Application	Non-Function Characteristics	Pictorial-model entity complement
'Change the satisfied weight setting for component mandate deployment'	A multiplication factor used to define the importance of the component mandate deployment optimisation score and overall influence of the score for the overall optimisation score and therefore the population search direction	Importance of component mandate deployment	NA
'Change the satisfied weight setting for assembly mandate deployment'	A multiplication factor used to define the importance of the assembly mandate deployment optimisation score and overall influence of the score for the overall optimisation score and therefore the population search direction	Importance of assembly mandate deployment	NA
'Change the satisfied weight setting for no colocation'	A multiplication factor used to define the importance of the component no-colocation mandate deployment optimisation score and overall influence of the score for the overall optimisation score and therefore the population search direction	Importance of component no-colocation mandate deployment	NA

CLI Prompt	Attribute Application	Non-Function Characteristics	Pictorial-model entity complement
'Change the NOT satisfied weight setting for connection latency'	This weight is used in the same way as the satisfied weight above but is separate to allow the modeller and optimisation process to account for unique NOT satisfy deployment profile and allow for an additional influence on the overall optimisation score and therefore the population search direction. NB: This weight is only invoked when the Boolean flag is set to allow for deployment options to be counted that not do satisfy the requirements	Importance of connection latency NOT being satisfied, when associative Boolean flag has been set to allow deployment options that do not satisfy certain non-functional requirements.	NA

CLI Prompt	Attribute Application	Non-Function Characteristics	Pictorial-model entity complement
'Change the NOT satisfied weight setting for component mandate deployment'	This weight is used in the same way as the satisfied weight above but is separate to allow the modeller and optimisation process to account for unique NOT satisfy deployment profile and allow for an additional influence on the overall optimisation score and therefore the population search direction. NB: This weight is only invoked when the Boolean flag is set to allow for deployment options to be counted that not do satisfy the requirements.	Importance of component mandate deployment NOT being satisfied, when associative Boolean flag has been set to allow deployment options that do not satisfy certain non-functional requirements.	NA
'Change the NOT satisfied weight setting for assembly mandate deployment'	This weight is used in the same way as the satisfied weight above but is separate to allow the modeller and optimisation process to account for unique NOT satisfy deployment profile and allow for an additional influence on the overall optimisation score and therefore the population search direction. NB: This weight is only invoked when the Boolean flag is set to allow for deployment options to be counted that not do satisfy the requirements.	Importance of assembly mandate deployment NOT being satisfied, when associative Boolean flag has been set to allow deployment options that do not satisfy certain non-functional requirements.	NA

CLI Prompt	Attribute Application	Non-Function Characteristics	Pictorial-model entity complement
'Change the NOT satisfied setting for no colocation'	This weight is used in the same way as the satisfied weight above but is separate to allow the modeller and optimisation process to account for unique NOT satisfy deployment profile and allow for an additional influence on the overall optimisation score and therefore the population search direction. NB: This weight is only invoked when the Boolean flag is set to allow for deployment options to be counted that not do satisfy the requirements.	Importance of component no colocation mandate deployment NOT being satisfied, when associative Boolean flag has been set to allow deployment options that do not satisfy certain non-functional requirements.	NA
'Change the 'n' number'	Used to set the value of the 'n' variable that is used within the behaviour workload modelling entities when functions are used to set levels of resource consumption within those workers	Used to set the resource utilisation in conjunction with a workload function	Used in conjunction with the complexity function and worker model entities within the Behaviour context window.

CLI Prompt	Attribute Application	Non-Function Characteristics	Pictorial-model entity complement
'Change the 'm' number'	Used to set the value of the 'm' variable that is used within the behaviour workload modelling entities when functions are used to set levels of resource consumption within those workers	Used to set the resource utilisation in conjunction with a workload function	Used in conjunction with the complexity function and worker model entities within the Behaviour context window.
'Change the 'f' number'	Used to set the value of the 'f' variable that is used within the behaviour workload modelling entities when functions are used to set levels of resource consumption within those workers	Used to set the resource utilisation in conjunction with a workload function	Used in conjunction with the complexity function and worker model entities within the Behaviour context window.

A.3 Text-based Workload Growth Modelling Elements

The listing below details the CLI options for changing the ‘n’, ‘m’ and ‘f’ variables that are utilised for complexity functions associated with CPU and memory utilisation, as well as bandwidth consumption for interface connections. The growth options are:

- ‘linear growth rate’
- ‘exponential growth’
- ‘half exponential growth rate’
- ‘quarter exponential growth rate’
- ‘Please enter the number of variable adjustments’
- ‘Please enter the size of change to be applied for each adjustment’

A.4 Text-based EC Modelling Elements

CLI Prompt	Description	Characteristic
'Change the size of EC population'	Used to set the size of the population for each generation created, which include both the local search processing and the global search processing detailed in the next section.	Size of the population from the solution space to be carried across the generation.
'Change the number of EC local Search'	Used to set the number of local searches to occur before the initial population is moved to the global search processing	How many times the local search process is applied to the initial population.
'Change the EC type of local search to 'exchange/inverse/jump'	Used to set the type of predefined local search processing to occur on the initial population. Details on these options found in next section	The optimisation algorithm chosen for processing the initial population
'Change the EC number base to 'integer/binary'	Used to set the base number used for the processing and search. Can be set to either an integer-based search process or binary-based search process. More provided in next section	Determines whether binary or integer base numbers are to be used for the processing.
'Change the EC number of generations'	This option sets the number of generations to be created from the global optimisation process.	times the global optimisation algorithm is executed to lead to the creation of the final population.

B. Evolutionary Computation XML

B.1 Computing Environment Instrumented Data

```
<nodes>
    <node cpu_hz="3.4700 GHz" cpu_mwips="2348.553526" cpu_name="Intel(R) Xeon(R)
    CPU X5677  @ 3.47GHz" cpu_vendor="GenuineIntel" ip="192.168.111.87"
    name="cranberry01" ram_availiable="3238" ram_free="3211" ram_total="3790">
        <tcp_latency ip="192.168.111.243">86</tcp_latency>
        <tcp_latency ip="192.168.111.97">151</tcp_latency>
        <tcp_latency ip="192.168.111.85">107</tcp_latency>
    </node>
    <node cpu_hz="3.4700 GHz" cpu_mwips="2229.858248" cpu_name="Intel(R) Xeon(R)
    CPU X5677  @ 3.47GHz" cpu_vendor="GenuineIntel" ip="192.168.111.243"
    name="cranberry02" ram_availiable="3268" ram_free="3288" ram_total="3790">
        <tcp_latency ip="192.168.111.97">85</tcp_latency>
        <tcp_latency ip="192.168.111.85">150</tcp_latency>
        <tcp_latency ip="192.168.111.87">101</tcp_latency>
    </node>
    <node cpu_hz="3.4700 GHz" cpu_mwips="2288.59661" cpu_name="Intel(R) Xeon(R)
    CPU X5677  @ 3.47GHz" cpu_vendor="GenuineIntel" ip="192.168.111.85"
    name="mandarin01" ram_availiable="3408" ram_free="2973" ram_total="3945">
    <tcp_latency ip="192.168.111.243">103</tcp_latency>
    <tcp_latency ip="192.168.111.97">117</tcp_latency>
    <tcp_latency ip="192.168.111.87">155</tcp_latency>
    </node>
    <node cpu_hz="3.4700 GHz" cpu_mwips="2579.220766" cpu_name="Intel(R) Xeon(R)
    CPU X5677  @ 3.47GHz" cpu_vendor="GenuineIntel" ip="192.168.111.97"
    name="mandarin02" ram_availiable="3454" ram_free="3364" ram_total="3945">
```

```
    <tcp_latency ip="192.168.111.243">145</tcp_latency>
    <tcp_latency ip="192.168.111.85">107</tcp_latency>
    <tcp_latency ip="192.168.111.87">108</tcp_latency>
    </node>
</nodes>
```

B.2 Global XML Details

```
<GlobalSettings id="2">
<GraphML_FileName id="3">
    base_model_test_v3.3.3_noDeployModelling_EC_Testing_noConstraints
</GraphML_FileName>
<Variables id="10">
    <NNumber id="11">1</NNumber>
    <MNumber id="12">1</MNumber>
    <FNumber id="13">1</FNumber>
</Variables>
<BitSize id="14">
    <Integer id="15">32</Integer>
    <Double id="16">64</Double>
</BitSize>
<Weights id="17">
    <ExecutionThreadWeight id="18">
            <Factor id="19">1</Factor>
    </ExecutionThreadWeight>
    <ConnectionConstraint id="20">
            <MandateAll id="21">True</MandateAll>
            <FactorSatisfied id="22">1</FactorSatisfied>
            <FactorNotSatisfied id="23">1</FactorNotSatisfied>
        </ConnectionConstraint>
        <ComponentDeploymentConstraint id="24">
            <MandateAll id="25">True</MandateAll>
            <FactorSatisfied id="26">1</FactorSatisfied>
            <FactorNotSatisfied id="27">1</FactorNotSatisfied>
        </ComponentDeploymentConstraint>
```

```
        <AssemblyDeploymentConstraint id="28">
            <MandateAll id="29">True</MandateAll>
            <FactorSatisfied id="30">1</FactorSatisfied>
            <FactorNotSatisfied id="31">1</FactorNotSatisfied>
        </AssemblyDeploymentConstraint>
        <NoColocationConstraint id="32">
            <MandateAll id="33">True</MandateAll>
            <FactorSatisfied id="34">1</FactorSatisfied>
            <FactorNotSatisfied id="35">1</FactorNotSatisfied>
        </NoColocationConstraint>
    </Weights>
    <EC_Detail id="36">
        <PopulationSize id="37">10</PopulationSize>
        <LocalSearchNumber id="38">2</LocalSearchNumber>
        <LocalSearchType id="39">exchange</LocalSearchType>
        <NumberBase id="40">integer</NumberBase>
        <GenerationsCreated id="41">20</GenerationsCreated>
    </EC_Detail>
</GlobalSettings>
```

B.3 Hardware XML Details

```
<HardwareDetail id="4">
<Nodes id="5">
<Node id="M400">
<NodeName id="42">cranberry01</NodeName>
<NodeMemory id="43">3211</NodeMemory>
<NodeCPU id="44">Intel(R) Xeon(R) CPU           X5677  @ 3.47GHz</NodeCPU>
<NodeCPUHz id="45">3.4700 GHz</NodeCPUHz>
<NodeCPUVendor id="46">GenuineIntel</NodeCPUVendor>
<NodeCPUMWIPS id="47">2348.553526</NodeCPUMWIPS>
<NodeOS id="48">Linux</NodeOS>
<OSBits id="49">64</OSBits>
<IPAddress id="50">192.168.111.87</IPAddress>
<ConnectionSpeedMB id="51">1000</ConnectionSpeedMB>
```

```
<SpareCapacityCPU id="52">0</SpareCapacityCPU>
<SpareCapacityMemory id="53">0</SpareCapacityMemory>
<WeightSpareCapacityCPU id="54">1</WeightSpareCapacityCPU>
<WeightSpareCapacityMemory id="55">1</WeightSpareCapacityMemory>
<Services id="56">
<Service id="57">SEM</Service>
</Services>
</Node>
<Node id="M401">
<Node id="M403">
<Node id="M404">
</Nodes>
<NodeConnection id="106">
<NodeOne id="107" referenceId="M400">192.168.111.87</NodeOne>
<NodeTwo id="108" referenceId="M401">192.168.111.243</NodeTwo>
<Latency id="109">86</Latency>
</NodeConnection>
<NodeConnection id="110">
<NodeConnection id="114">
<NodeConnection id="118">
<NodeConnection id="122">
<NodeConnection id="126">
<NodeConnection id="130">
<NodeConnection id="134">
<NodeConnection id="138">
<NodeConnection id="142">
<NodeConnection id="146">
<NodeConnection id="150">
</HardwareDetail>
```

B.4 Software XML Details

```
<SoftwareDetail id="6">
    <Assemblies id="7">
        <Assembly id="M910">
```

```
            <Component id="M911">
                <ComponentName id="154">Sender_1</ComponentName>
                <Interface id="M919">
                    <InterfaceName id="155">txMessage_1</InterfaceName>
                    <WorkloadPair id="156">
                        <CPUWorkload id="157">
                            <Function id="158">225</Function>
                        </CPUWorkload>
                        <MemoryWorkload id="159">
                            <Function id="160">3000</Function>
                        </MemoryWorkload>
                        <ForLoop id="161">10</ForLoop>
                    </WorkloadPair>
                    <WorkloadDataType id="162">Double</WorkloadDataType>
                </Interface>
            </Component>
            <Component id="M979">
            <Component id="M1038">
            <Component id="M1099">
            <Component id="M386">
            <Component id="M654">
        </Assembly>
        <Assembly id="M1154">
    </Assemblies>
    <ExecutionThread id="8">
        <CriticalConnection id="M1583">
            <ConnectionSrc id="348" interfaceId="M1033"/>
            <ConnectionDst id="349" interfaceId="M1279"/>
            <InternalConnection id="350">false</InternalConnection>
        </CriticalConnection>
        <CriticalConnection id="M1582">
        <CriticalConnection id="M1581">
        <CriticalConnection id="M1486">
        <CriticalConnection id="M1485">
    </ExecutionThread>
    <SystemConnections id="9">
        <SystemConnection id="M1573">
            <ConnectionSrc id="308" interfaceId="M1033"/>
```

```
            <ConnectionDst id="309" interfaceId="M1279"/>
            <ConnectionType id="310">ZMQ</ConnectionType>
            <ConnectionPayload id="311">50xm</ConnectionPayload>
            <ConnectionFrequency id="312">f</ConnectionFrequency>
        </SystemConnection>
        <SystemConnection id="M1571">
        <SystemConnection id="M1568">
        <SystemConnection id="M1566">
        <SystemConnection id="M1565">
        <SystemConnection id="M1564">
        <SystemConnection id="M1562">
        <SystemConnection id="M1561">
    </SystemConnections>
</SoftwareDetail>
```

C. MEDEA Modelling Flow

C.1 Information Modelling Component Flow

The following is a list of non-functional requirements and constraints that feed into the deployment optimisation search process:

1. Critical thread latency:
   Refers to the identification of an end-to-end thread or path within the system design that would be seen as a critical design element central to the functionality and performance of the overall system. The temporal performance of this thread is minimised as much as possible, while still working to satisfy other non-functional requirements and constraints.
   The definition of this critical thread consists of identifying the software components, component workload, component interface, intra-component connections and inter-component connections (including delegate inter-component connections between assemblies) that exist within a path of interest within the overall system design.
   In this first instance, the deployment optimisation will only consider a single design thread of interest. Future concepts may look into the potential for deployment around multiple design threads or paths.
2. String latency:
   Allows for a requirement that a number of connected interfaces (one or more but less than the critical thread count) found within the critical thread have a total latency that does not exceed a maximum latency. As a result, the optimisation process will seek to minimise a critical thread's latency, while still ensuring the identified string total latency is equal to or less than the defined maximum latency.
3. Interface latency:
   Requires the optimisation processing to consider the maximum latency time of a single interface within the critical thread. Once again, the optimisation

algorithm will aim to minimise a critical thread's latency, while still ensuring the identified interface latency is equal to or less than the maximum latency time defined.

4. CPU and memory spare capacity:
   The definition allows the modeller to define how much spare capacity they need for both CPU and memory. The optimisation processing will then aim to drive spare capacity for each resource on each computing node to be as little as possible for deployment options identified from the search, while aiming to ensure the spare capacities are maintained as much as possible.
5. Software component and assembly mandate deployment:
   The mandate deployment constraints, which include component collocations, and identified component and assembly deployment to particular computing nodes, guides the optimisation search process to only consider the deployment options population members that abide by those constraints.
   In addition to searching for deployments options that satisfy the mandate deployment requirements, the optimisation search process can be configured to consider population members that do not satisfy the requirements. In this configuration, deployment options can be explored when all or a subset of the deployment constraints are broken.

C.2 Critical Thread Temporal Performance Calculation Flow

Detailed description for each part of the execution flow are as follows:

1. **Node Deployment Profile:** The Objective Function Algorithm traces through every node defined within the graphML which represents the available computing environment for deployment. Once traced and the information extracted, the process uses that information and applies it the current solution search (deployment profile) where each software component (and expected workload) is assigned to a particular computing node.
2. **Profile CPU and Network Interface Utilisation:** Using the node deployment profiles for the available computing environment, total consumptions of CPU and network bandwidth resources are established for each computing node, for that particular search solution. For the CPU resources this is measured in MWIPS, while the network bandwidth is measured in Mbits/s.
3. **Computing Environment Resource Utilisation Profile Construction:** Us-

ing the output from the node CPU and network utilisation process, the results are translated into percentages and aggregated to produce a profile for the entire computing environment for that particular search solution.

4. **Critical Chain Trace and Extraction Process:** This process steps through each element of the critical thread defined within the System Optimisation Model. It steps through all the software component interfaces and associated workers, as well as whether the current critical thread segment is a 'intra' or 'inter' software component connection (which triggers additional calculations associated with non-functional constraints)
5. **CPU Utilisation Details Extraction:** CPU utilisation details for computing nodes associated with the critical chain are extracted from the 'Computing Environment Resource Utilisation Profile' created earlier.
6. **Network Utilisation Details Extraction:** Network utilisation details for computing nodes associated with the critical chain are extracted from the 'Computing Environment Resource Utilisation Profile' created earlier.
7. **Critical Chain Element Temporal Performance Calculation:** The execution flow for the time calculation can be found in Figure 5.17 and consists of:
    - **Connection Source Worker CPU Processing Time Calculation:** With all CPU processing workloads calculated around MWIPS the processing time is a straightforward ratio calculation based on the worker number of MWIPS to be executed, divided by the total available MWIPS resources available on the particular computing node. This ratio is then directly correlated to time needed to process the current workload in terms of seconds (NB: if the component interface has more than one workload associated with it via conditional logic, the EC algorithm will extract the details of the largest workload and use that for its optimisation search).
    - **Connection Destination Worker CPU Processing Time Calculation:** As per the Connection Source calculation above, the Connection Destination calculations are the same.
    - **Calculate CPU Utilisation Time Delay and Apply:** Using the CPU utilisation percentage established for the entire computing environment, each node utilisation percentage is then applied to the 'Utilisation versus Time Delay' performance curve (Figure 5.7) calculated previously. Using the results from the performance curve application, a delay multiplier is applied at various points within the critical thread segment time/latency calculation (Figure 5.17). This calculation is executed for

every component CPU workload and the entire set of calculations is repeated with every new search solution (deployment profile).

- **Payload Transmission CPU Processing Time Calculation for Source and Destination Connection End:** Using the details held within the XML performance files for payload transmission for ZeroMQ and interpolation, CPU workload (which is captured as a percentage of the CPU usage for each computing node CPU) is associated with the particular payload size transmissions, which are then extracted. Once again, based on percentage of utilisation of the CPU, the resulting time is a straight forward ratio calculation in seconds for the particular node.
- **Payload Transmission and Reception Latency Calculations:** As with processing requirements for ZeroMQ transmission, latency performance for ZeroMQ for payload sizes are also instrumented and recorded in XML files for each computing node. These sample points are then used with interpolation to determine transmission latency for a particular connection.
- **Calculate Network Bandwidth Utilisation Time Delay and Apply:** As with the CPU time delay calculation, the network bandwidth utilisation time delay is calculated by using the network utilisation percentage established for each node and applying it to the resource utilisation delay performance curve (Figure 5.7) calculated previously. It is then applied to the 'Calculate Network Bandwidth Utilisation Time Delay and Apply' part of the critical thread segment time/latency calculation execution flow (Figure 5.17).
- **Calculate Maximum Connection Latency:**Following the premise of this modelling and prediction paradigm research, all input performance data is based on the worst-case scenario. In this instance, the calculation of the latency for transmission for either end of the connection is calculated and the worst performing results of the connection in the optimisation algorithm. Transmission latency is calculated in milliseconds.

8. **Critical Thread Element Temporal Performance Aggregation:** All timing results calculated with associating resource utilisation delay factors are then aggregated to produce the latency for that section of the critical thread being investigated. This is then further aggregated with all the previous segment latency results to produce the overall critical thread end-to-end latency.

D. Verification Framework and Calibration Experiment Details

D.1 Verification Framework Message Set

- **Message header: 'INTER'**
  - The INTER text string indicates that this logging event has captured a flow of the critical thread that is between software components utilising the middleware and network connection available within the MEDEA runtime computing environment. It is the flow from the output port of a software component to the input port of another software component and both are identified as part of the critical path of the system design model.
  - The structure of this text string is:
    - "INTER, eventTime, %s, messageId, %i, messageCreationTime, %s, ECDetail_1, %s " where:
      - eventTime -> is the timestamp for this currently logged event
      - messageId -> unique identifier for the message being sent along the identified critical path
      - messageCreationTime -> the timestamp for when the current message was originally created within the 'sender' component
      - ECDetail_1 -> The text string capturing the critical path and non-functional performance requirement details
- **Message header: 'INTRA'**
  - The INTRA text string indicates that this logging event has captured the flow of the critical thread that is within a software component. This could be via an internal periodic event or the reception of a message via the input port of the software component. It indicates the flow is about to execute workloads associated with the input and output ports,

as well as any associating logic found within that software component.

- The structure of this text string is:
  - "INTRA, eventTime, %s, messageId, %i, messageCreationTime, %s, ECDetail_1, %s " where:
    - eventTime -> is the timestamp for this currently logged event
    - messageId -> unique identifier for the message being sent along the identified critical path
    - messageCreationTime -> the timestamp for when the current message was originally created within the 'sender' component
    - ECDetail_1 -> The text string capturing the critical path and non-functional performance requirement details

- *Message header:' ECDetail'*
  - The ECDetail text string captures the modelled information on a segment of the identified critical path. This includes details on the system model elements (graphML id information) that make up the particular segment, non-functional performance requirements and segment positioning details.
  - The structure of this text string and an example of this text string is below:
    - "criticalChain, 'X', 'SegmentPosition', portId_Tx, 'XXX', portId_Rx, 'XXX', internalConnection, 'X', maxLatency, 'XXX', maxLatencyStringGroup, 'X,XXXX, XXX'"
    - "criticalChain, y, START, portId_Tx, 919, portId_Rx, 1031, internalConnection, n, maxLatency, 100, maxLatencyStringGroup, 1, START, 650"
      where:
      - criticalChain -> indicates whether the component interface is part of the identified critical path. If the component interface is not part of the critical path, then the additional information following is not required to the added to the ECDetail text string. This will either be 'y' or 'n'.
      - 'SegmentPosition' -> is used to indicate where this particular segment sits within the identified critical path. This can either be 'START', 'Middle' or 'END'
      - portID_Tx -> the number following this label indicates the particular component interface that is the output port or transmitter of the identified segment. The number is based on the graphML id number of that particular output port.

- portID_Rx -> the number following this label indicates the particular component interface that is the input port or receiver of the identified segment. The number is based on the graphML id number of that particular input port.
- internalConnection -> this indicates whether the current segment is internal to a software component or whether the segment is a connection between software components. This will either be 'y' or 'n'. It should be noted that if the connection is internal to a software component, then the transmitter end of the segment will be the input port of the software component and the receiver end of the segment will be the output port of the software component.
- maxLatency -> this part of the text string indicates whether the current segment (and its output port) has a latency performance requirement associated with it. If there is no maximum latency time requirement then it becomes 'NA'.
- maxLatencyStringGroup -> in the same way a latency requirement is attached to a single segment of the identified thread, this label indicates whether the current segment is part of a string segment that has to satisfy a maximum latency requirement. There are two parts that make up this definition. The first is the unique identifier for the particular segment string of interest. The second part is the position of the segment within that currently identified string, and can be either 'START', 'Middle' or 'END'. In the case of a segment being the 'START' of the segment string, an additional piece of information is attached to the text string. This is a number indicating the maximum latency time required.

D.2 Predicted and Measured Scatter Plots

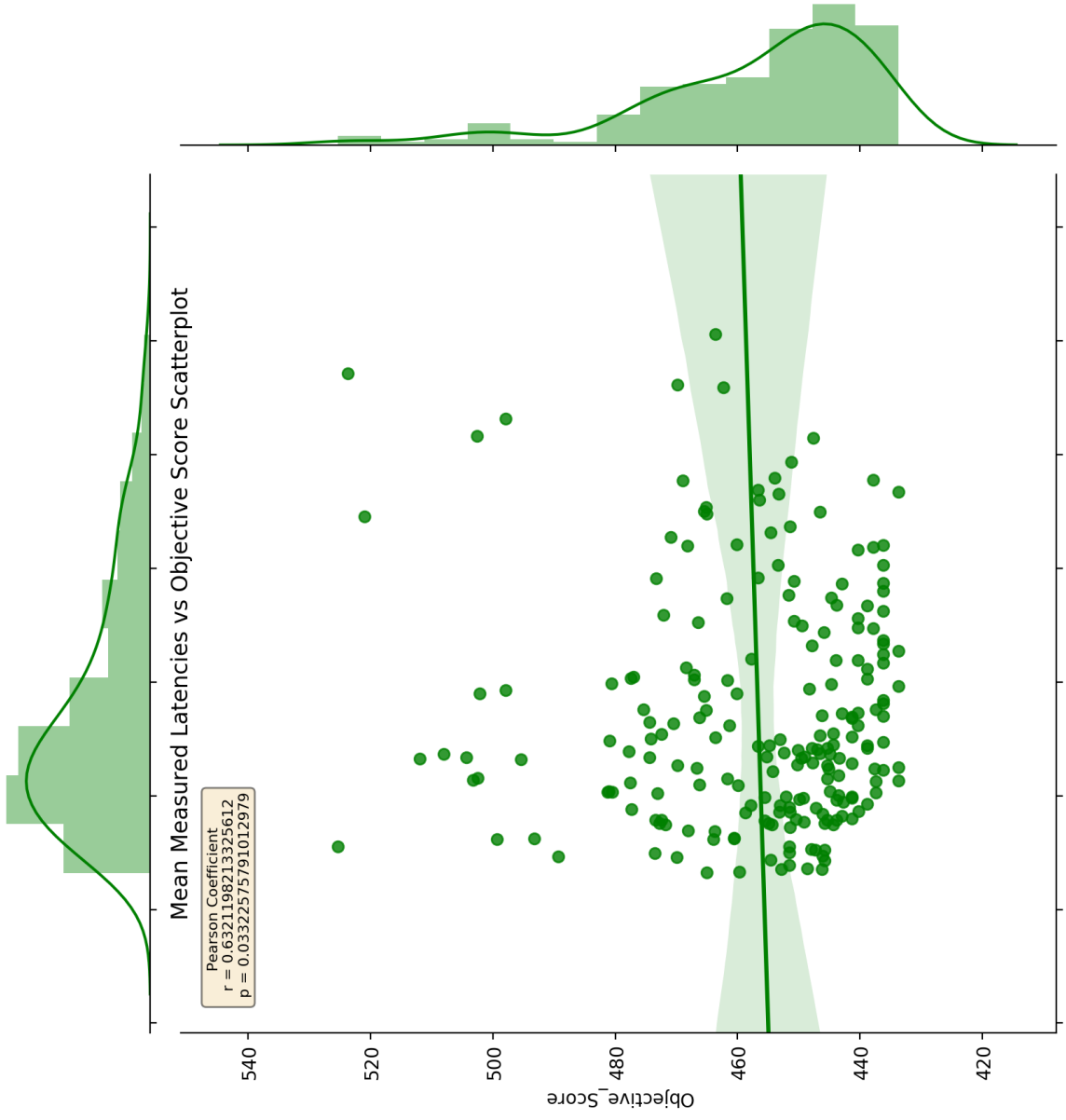

Figure D.1: 16Gb Network Bandwidth and Pearson Coefficient r=0.632

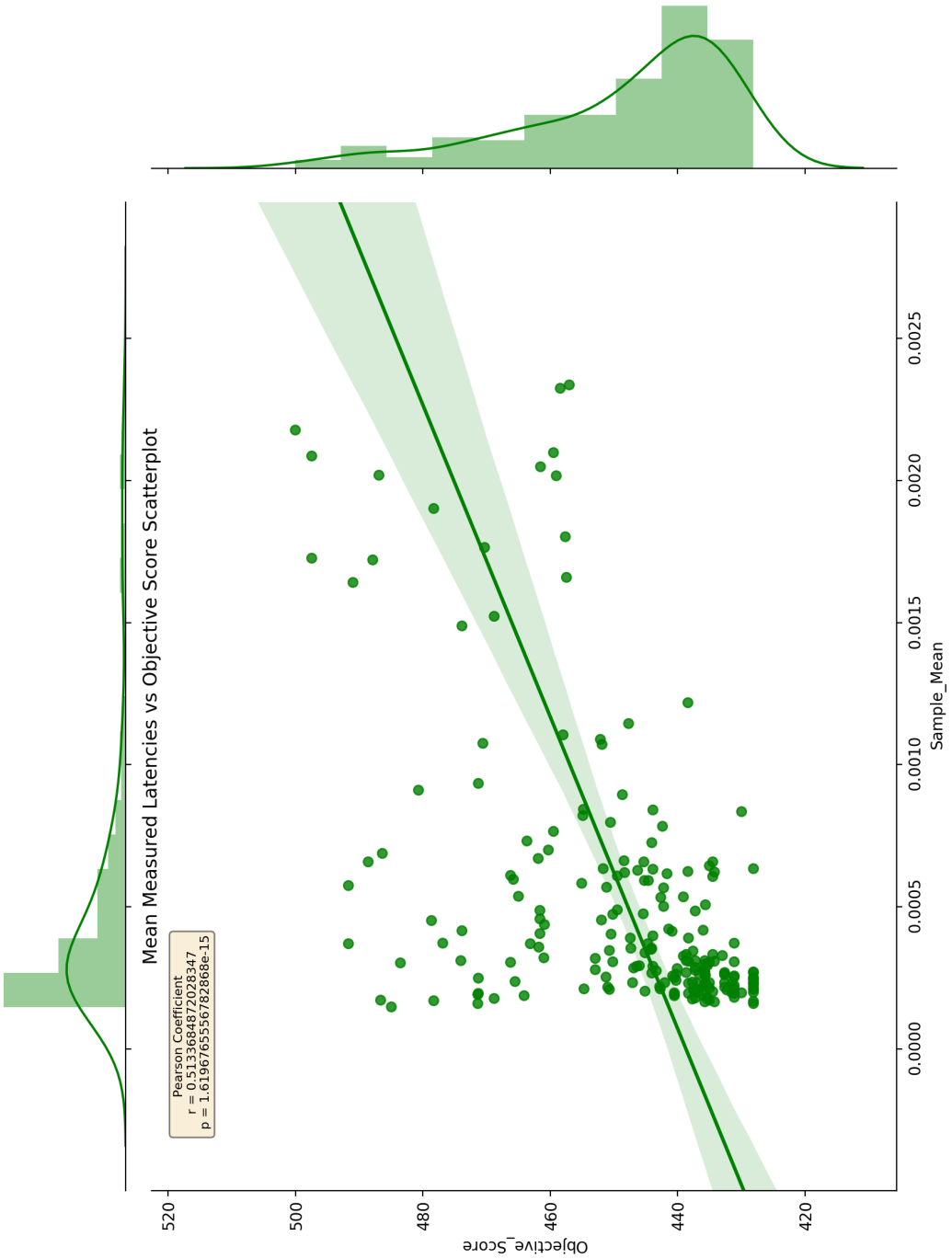

Figure D.2: 16Gb Network Bandwidth and Pearson Coefficient r=0.513

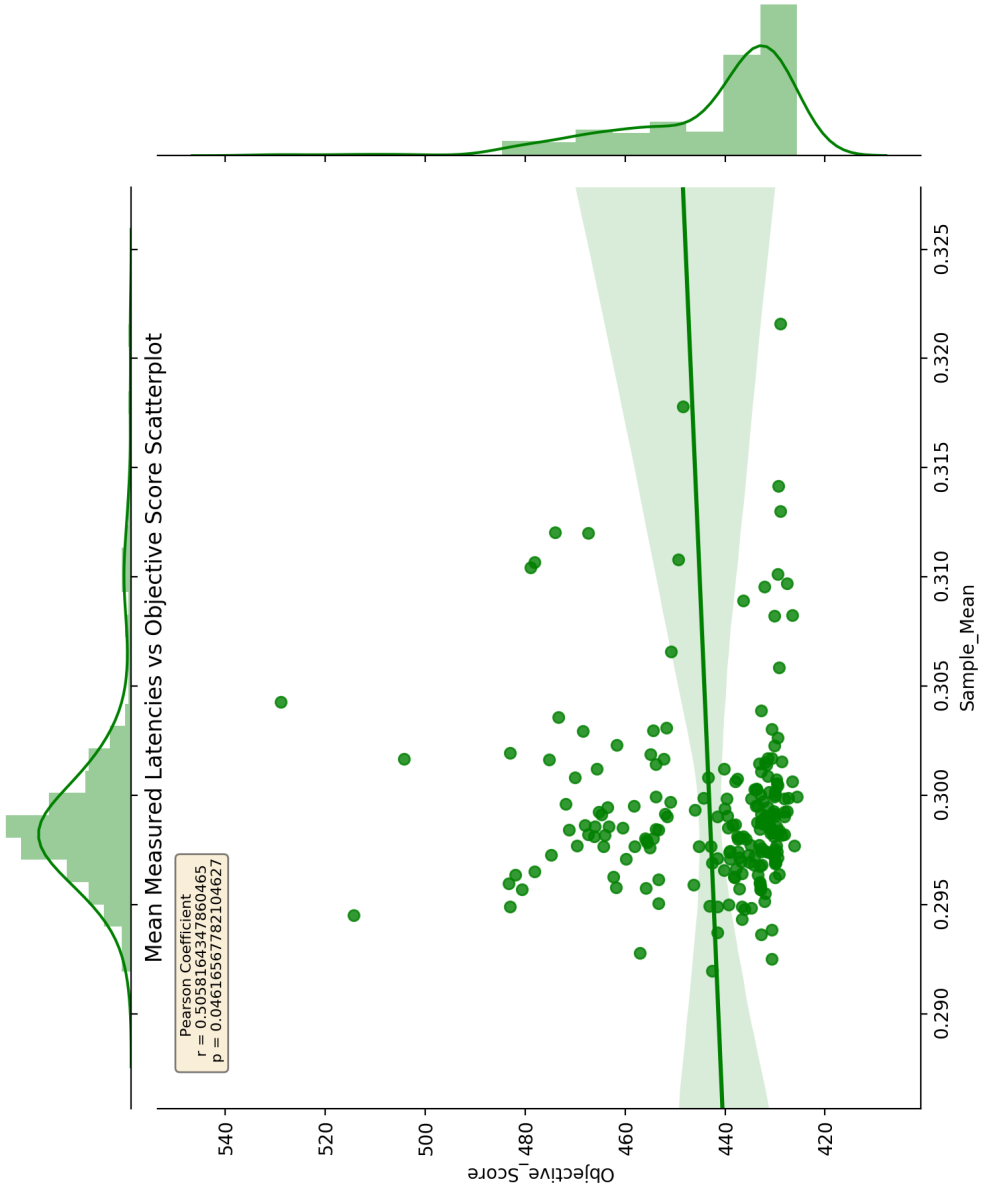

Figure D.3: 16Gb Network Bandwidth and Pearson Coefficient r=0.506

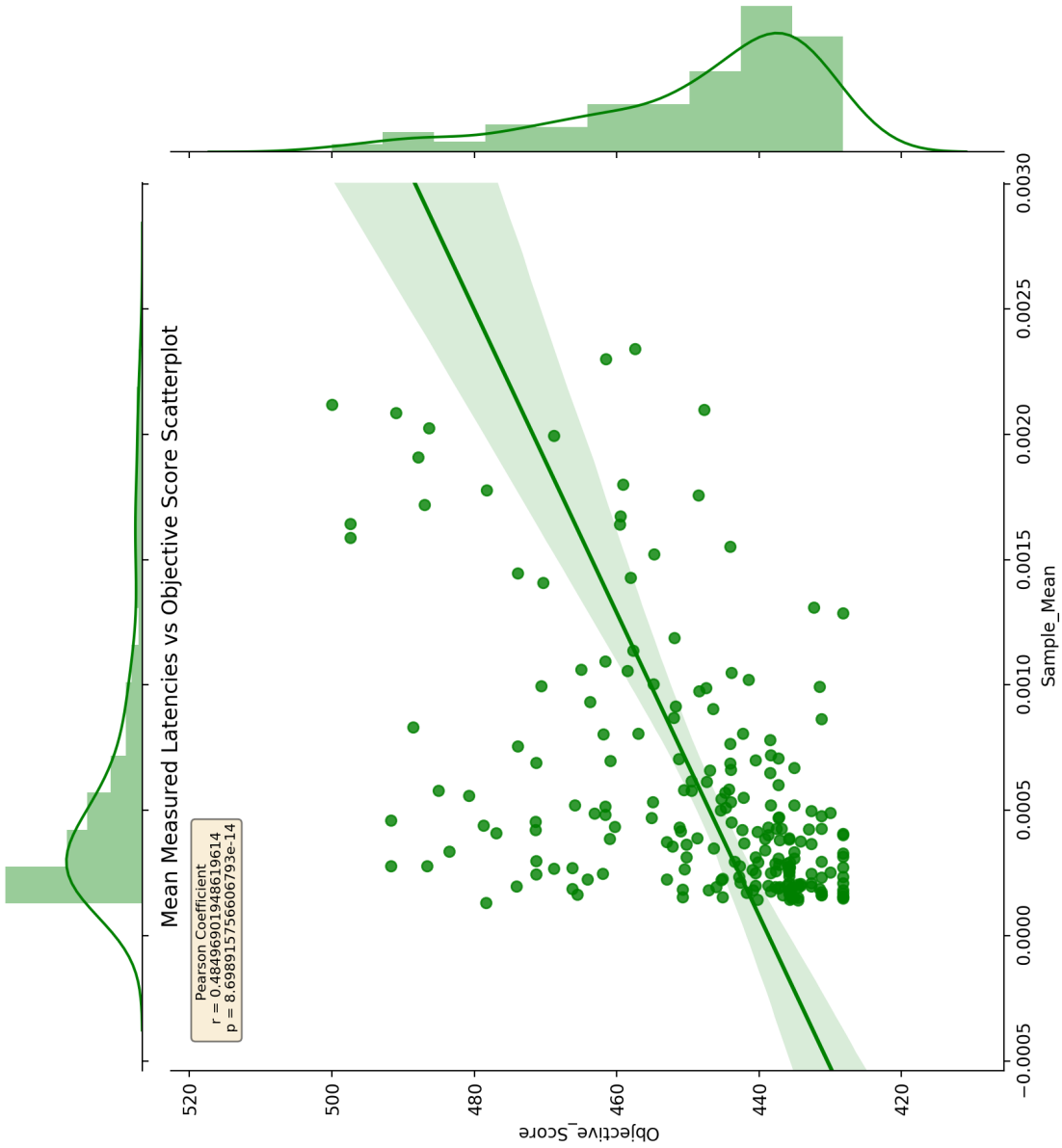

Figure D.4: 16Gb Network Bandwidth and Pearson Coefficient r=0.484

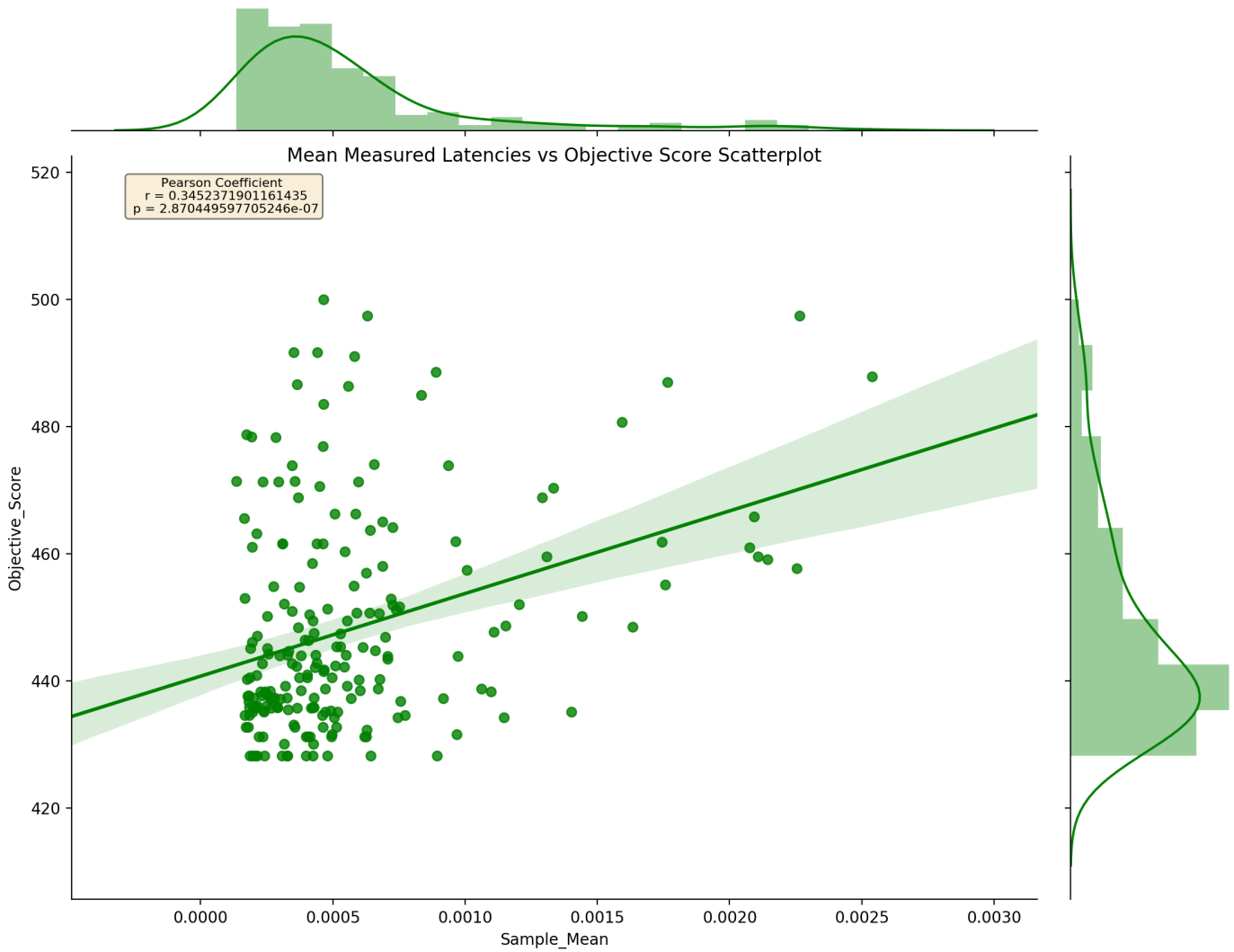

Figure D.5: 1Gb Network Bandwidth and Pearson Coefficient r=0.345

D.3 Latency Predicted and Measured Polynomial Fit Plots

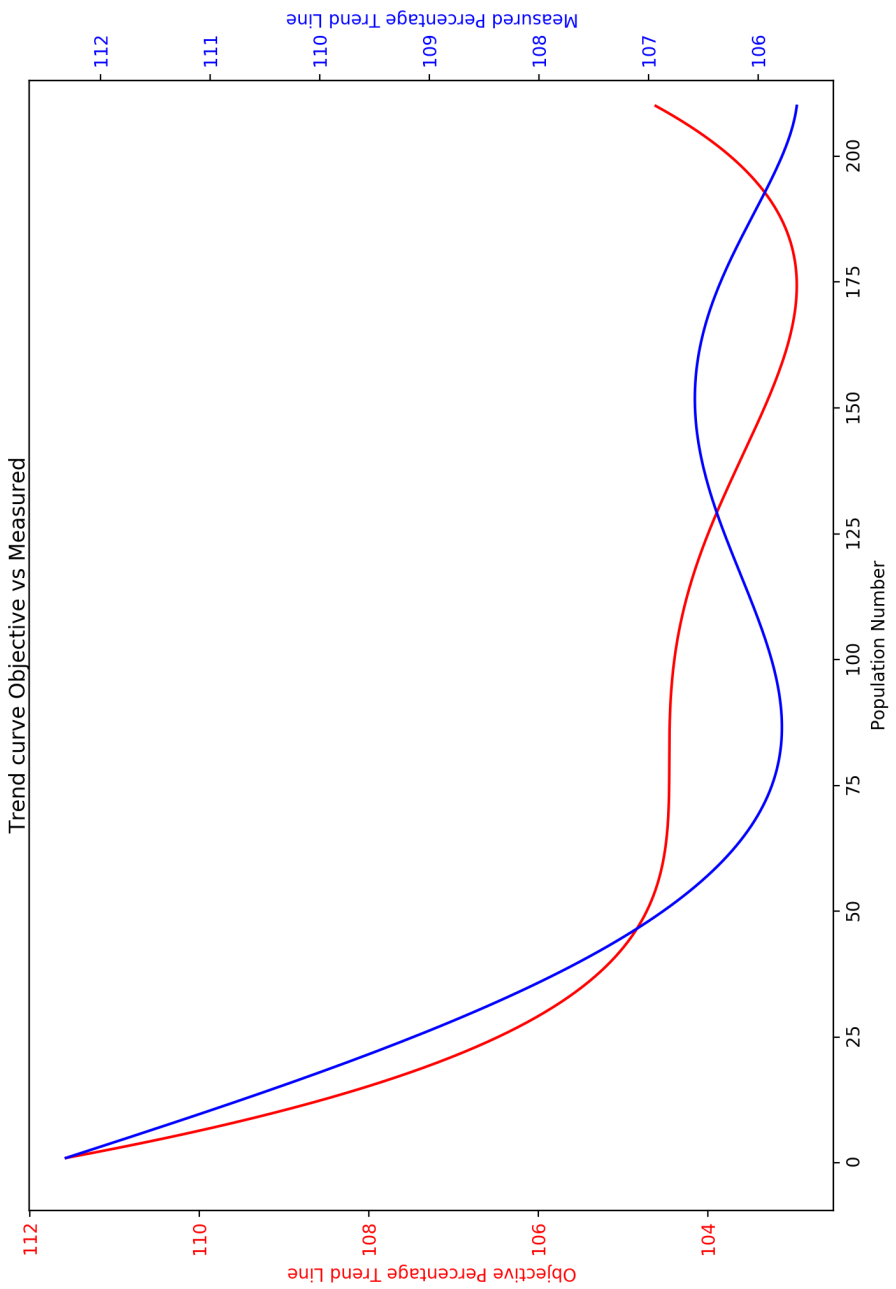

Figure D.6: 16Gb Network Bandwidth

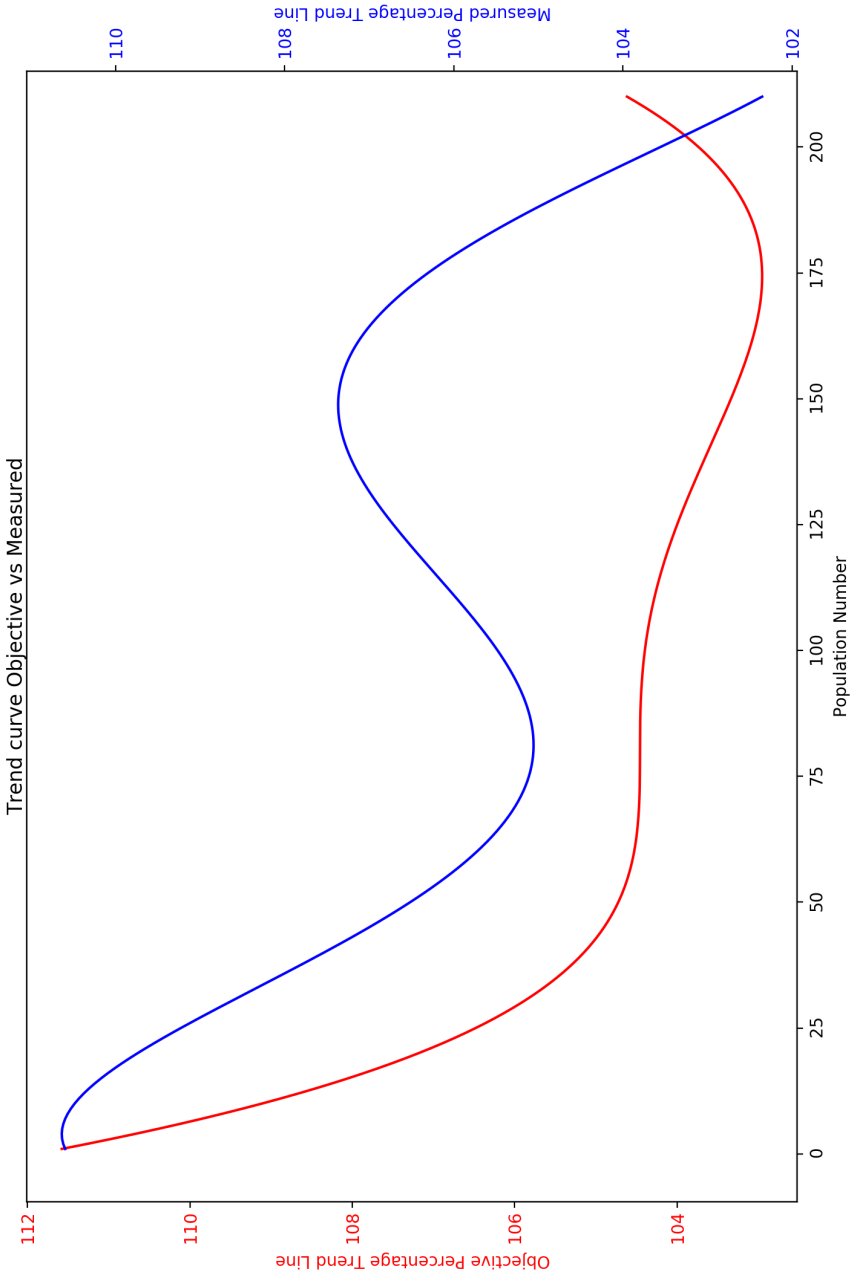

Figure D.7: 1Gb Network Bandwidth

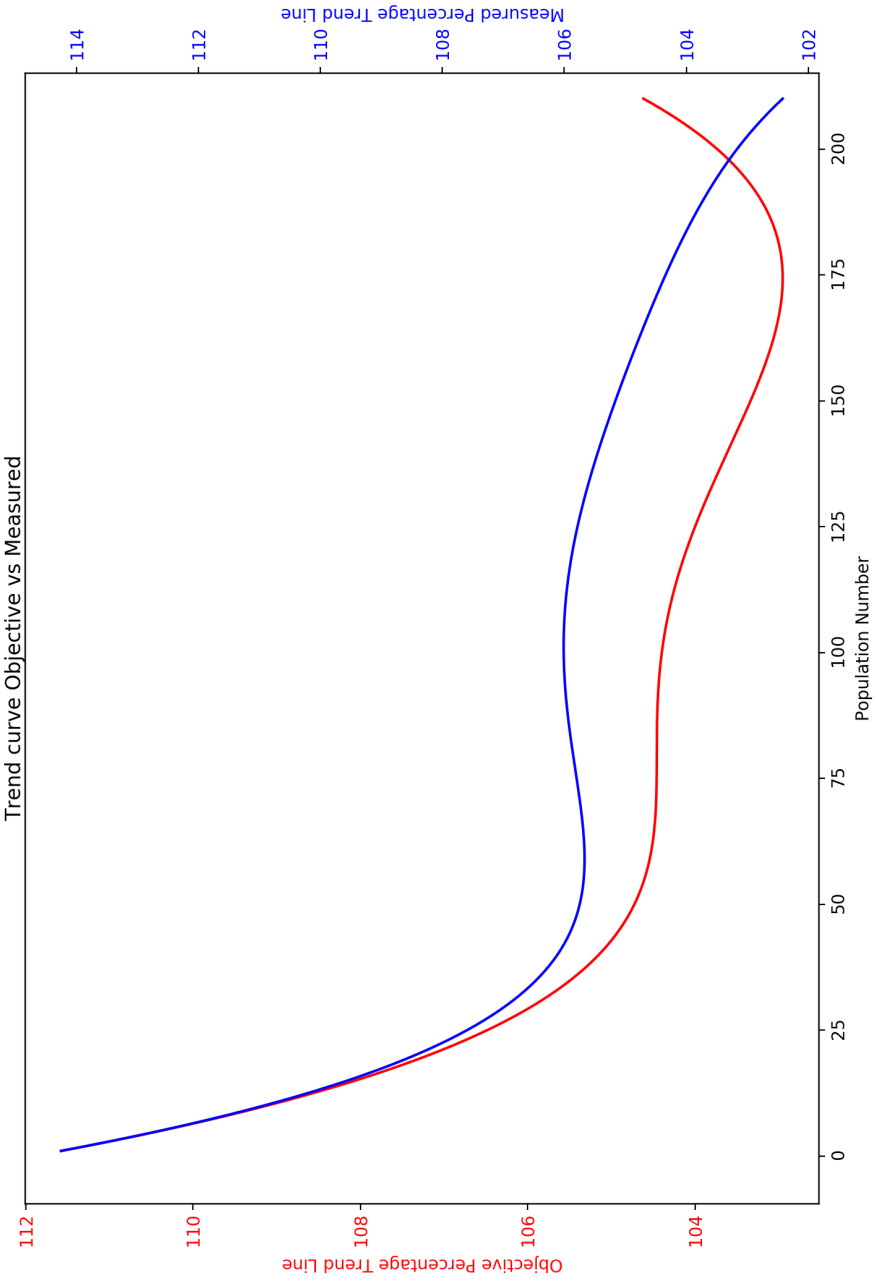

Figure D.8: 1Gb Network Bandwidth

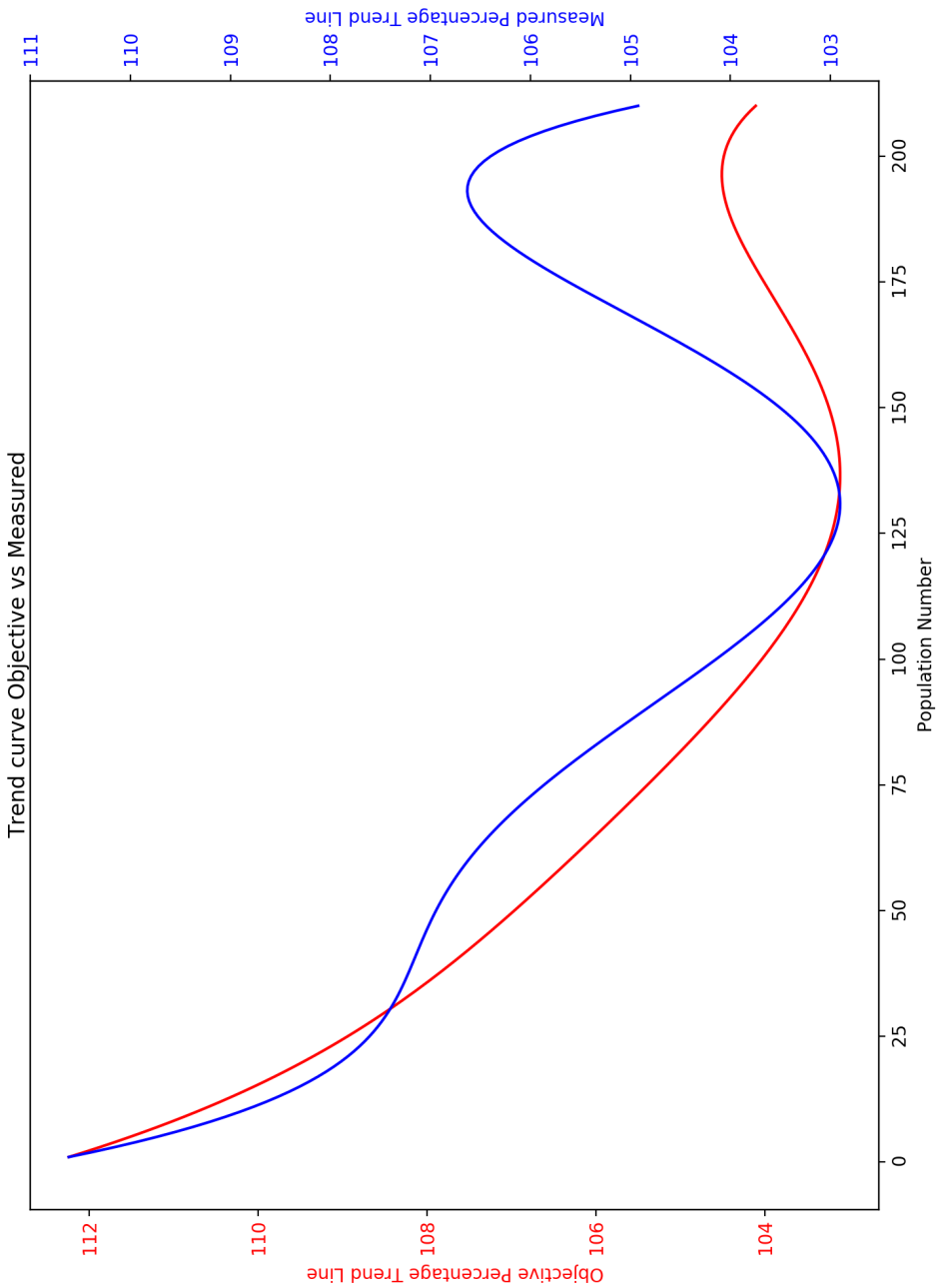

Figure D.9: 100Mb Network Bandwidth

D.4 Objective Function Diversity and Convergence

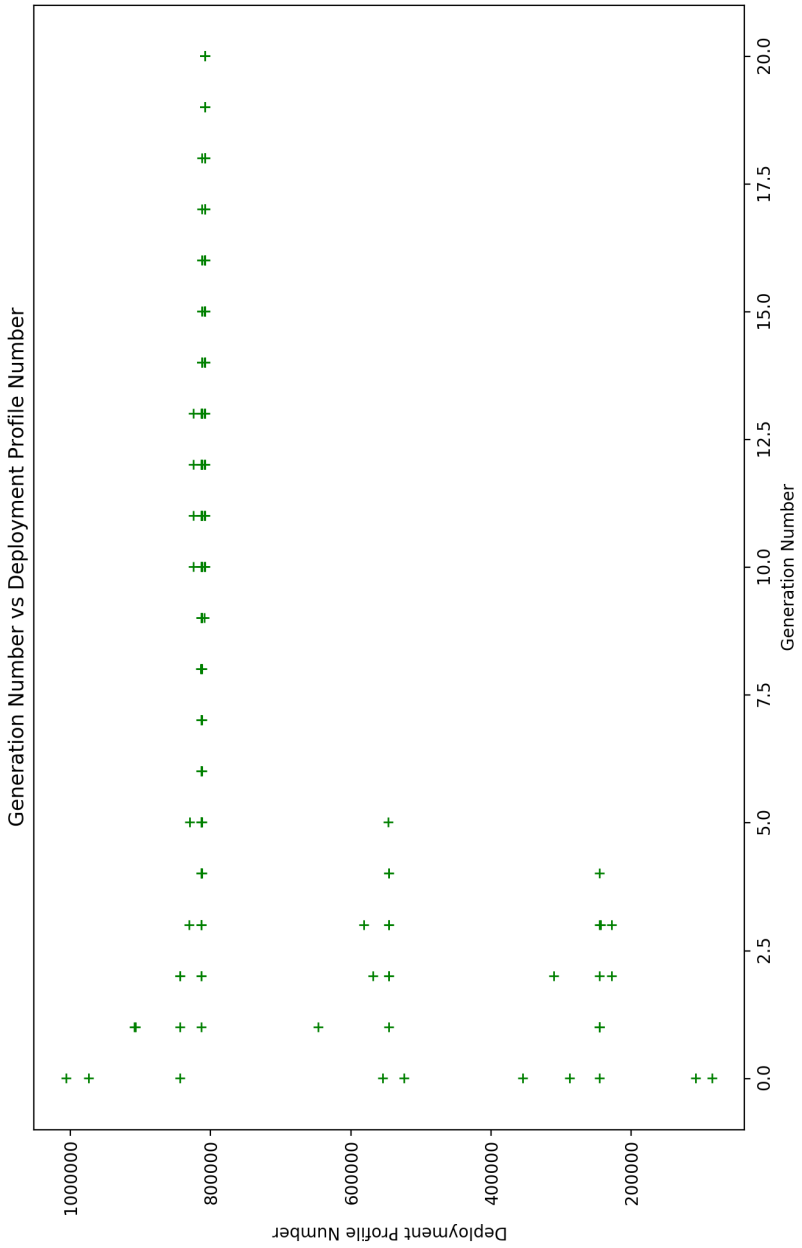

Figure D.10: Objective Function Test Results - 16Gb

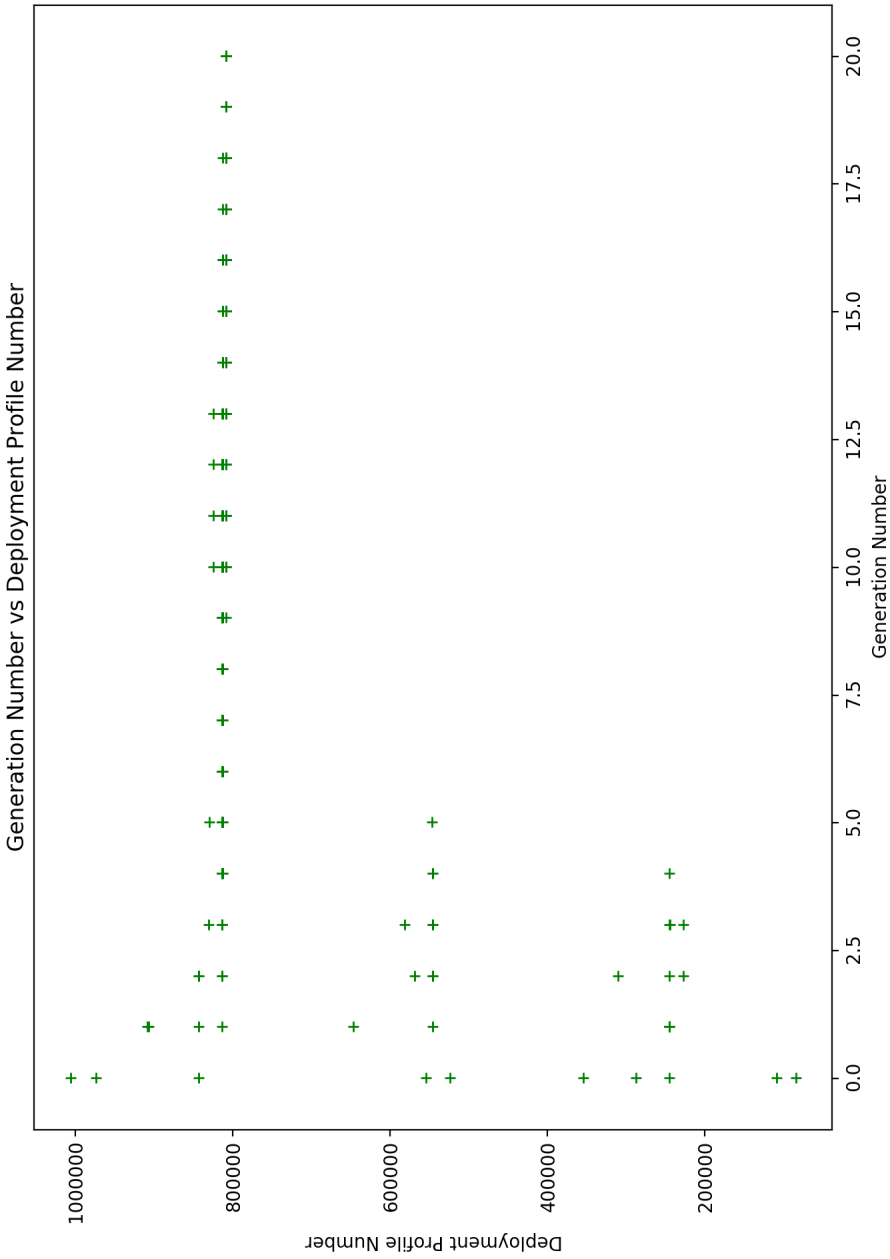

Figure D.11: Objective Function Test Results - 1Gb

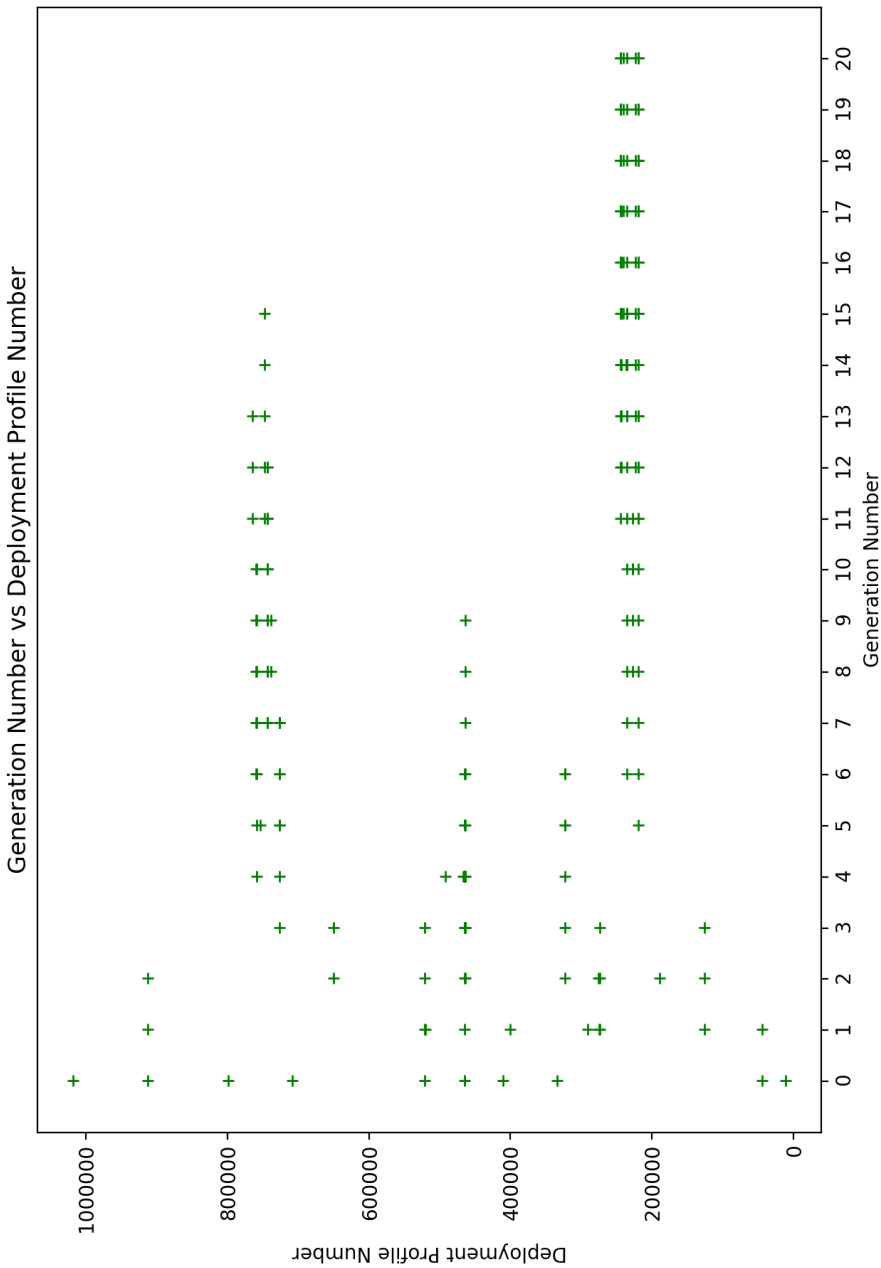

Figure D.12: Objective Function Test Results - 100Mb

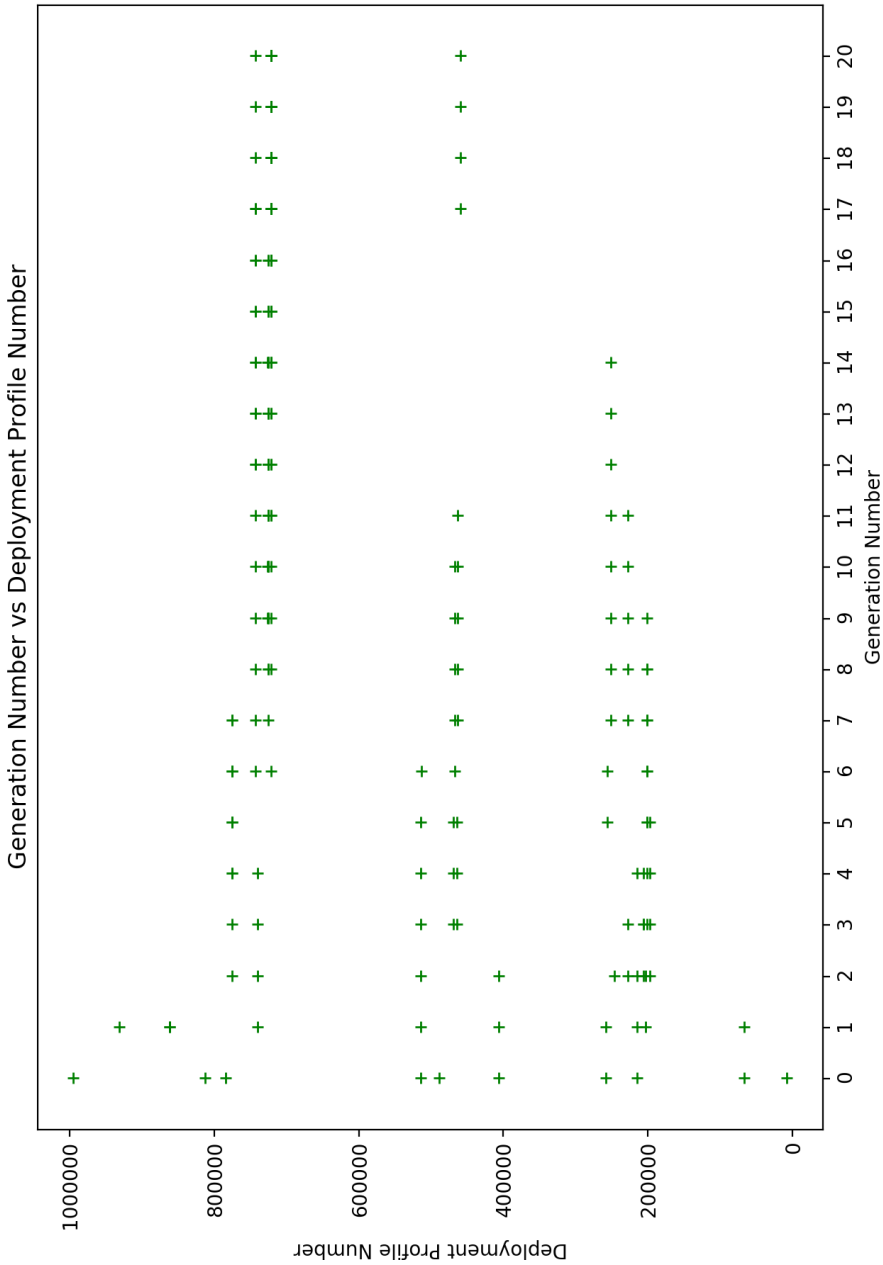

Figure D.13: Objective Function Test Results - 10Mb

D.5 Generation and Population Investigation Results

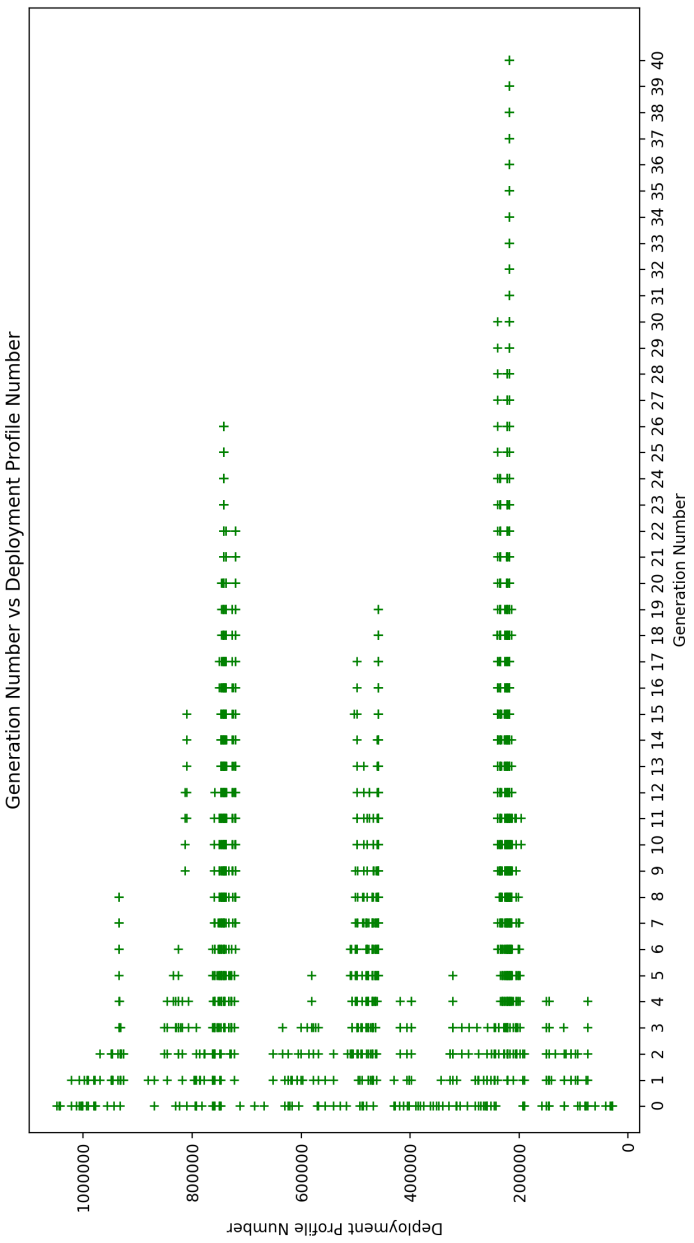

Figure D.14: Initialised Specialised Population and Number of Generations Test Results

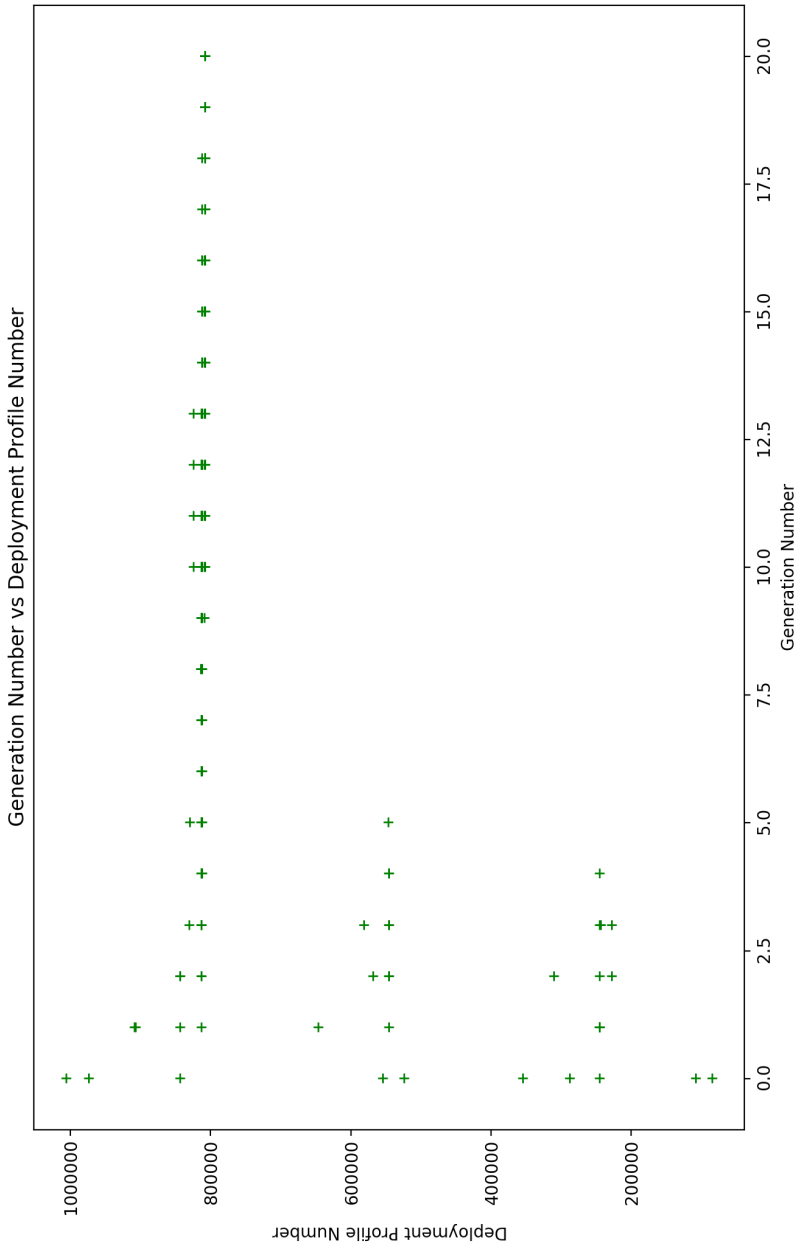

Figure D.15: Objective Function Test Results - 16Gb

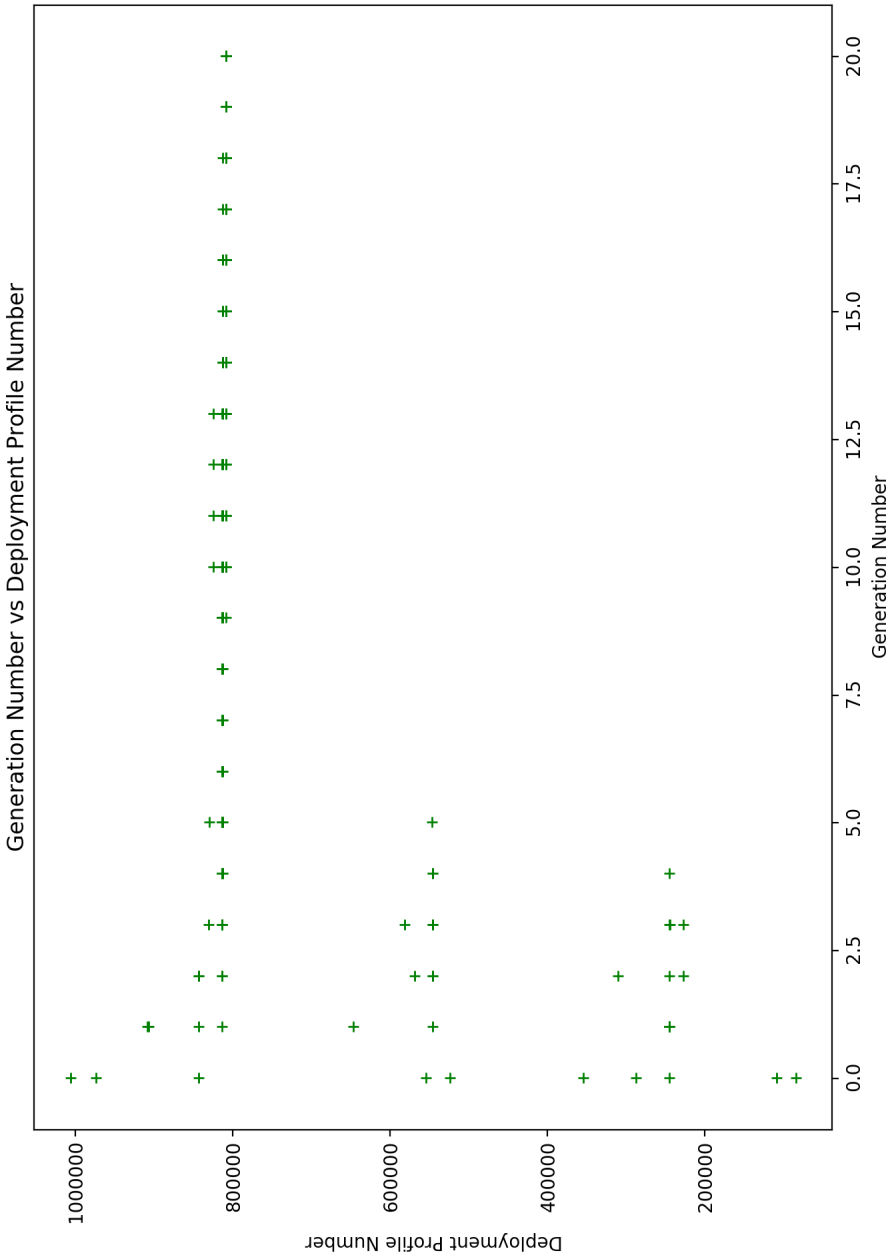

Figure D.16: Objective Function Test Results - 1Gb

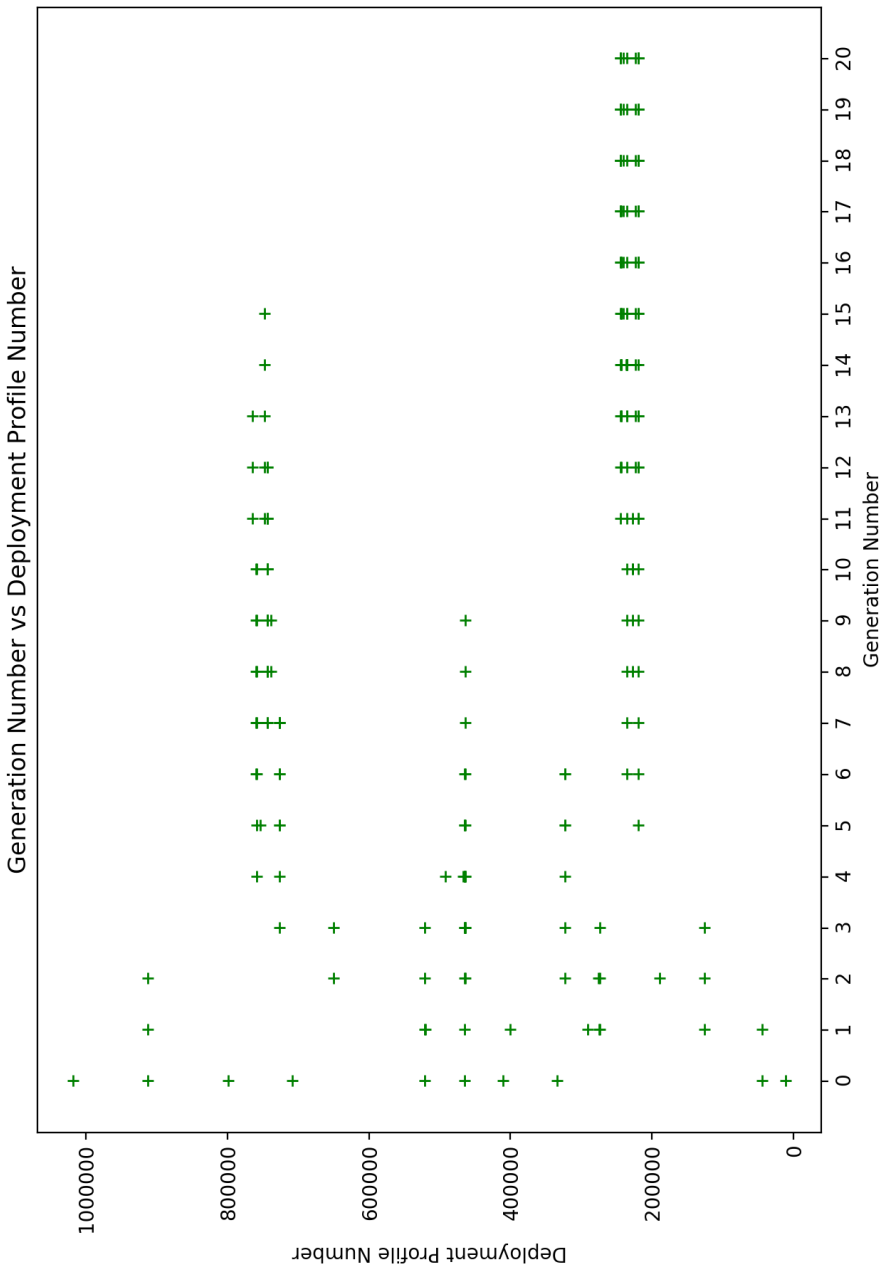

Figure D.17: Objective Function Test Results - 100Mb

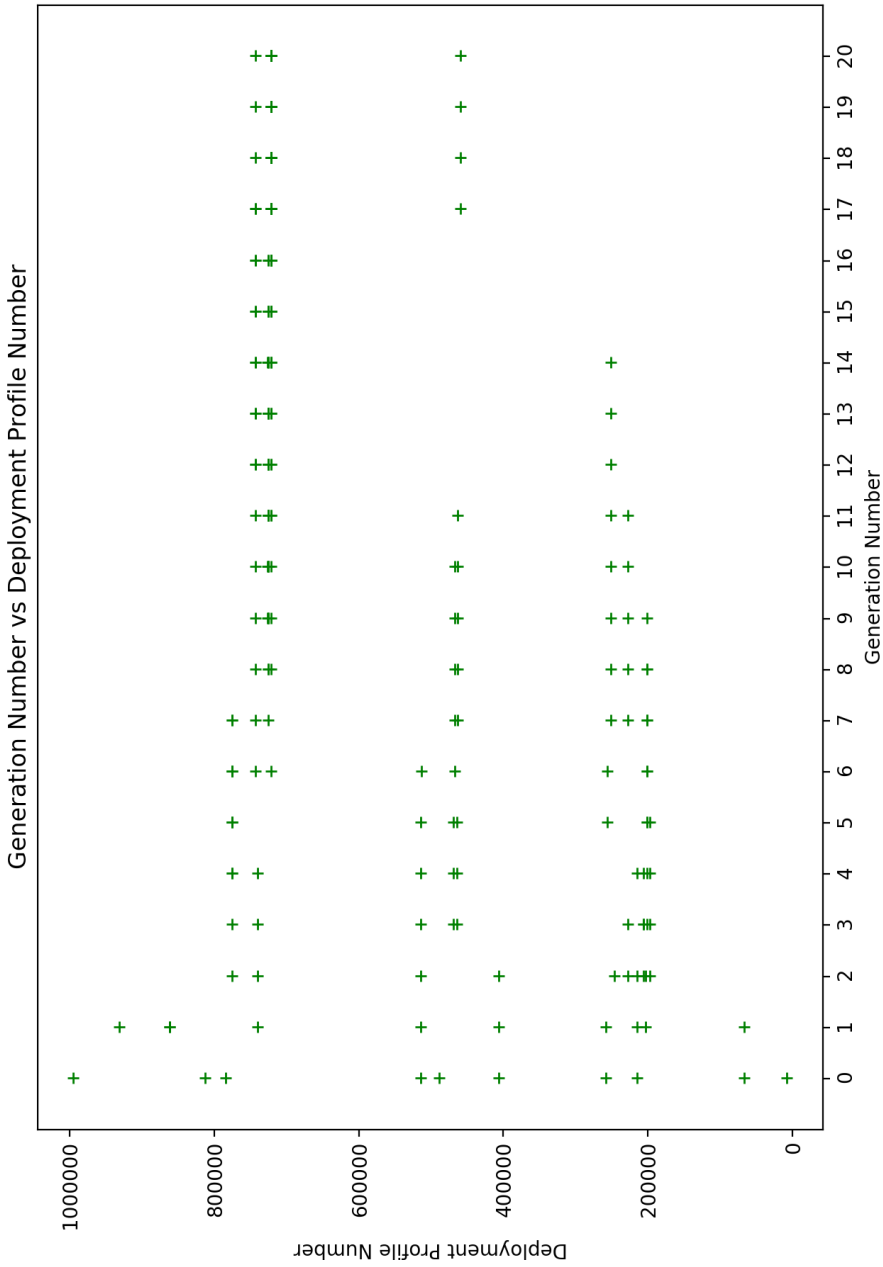

Figure D.18: Objective Function Test Results - 10Mb

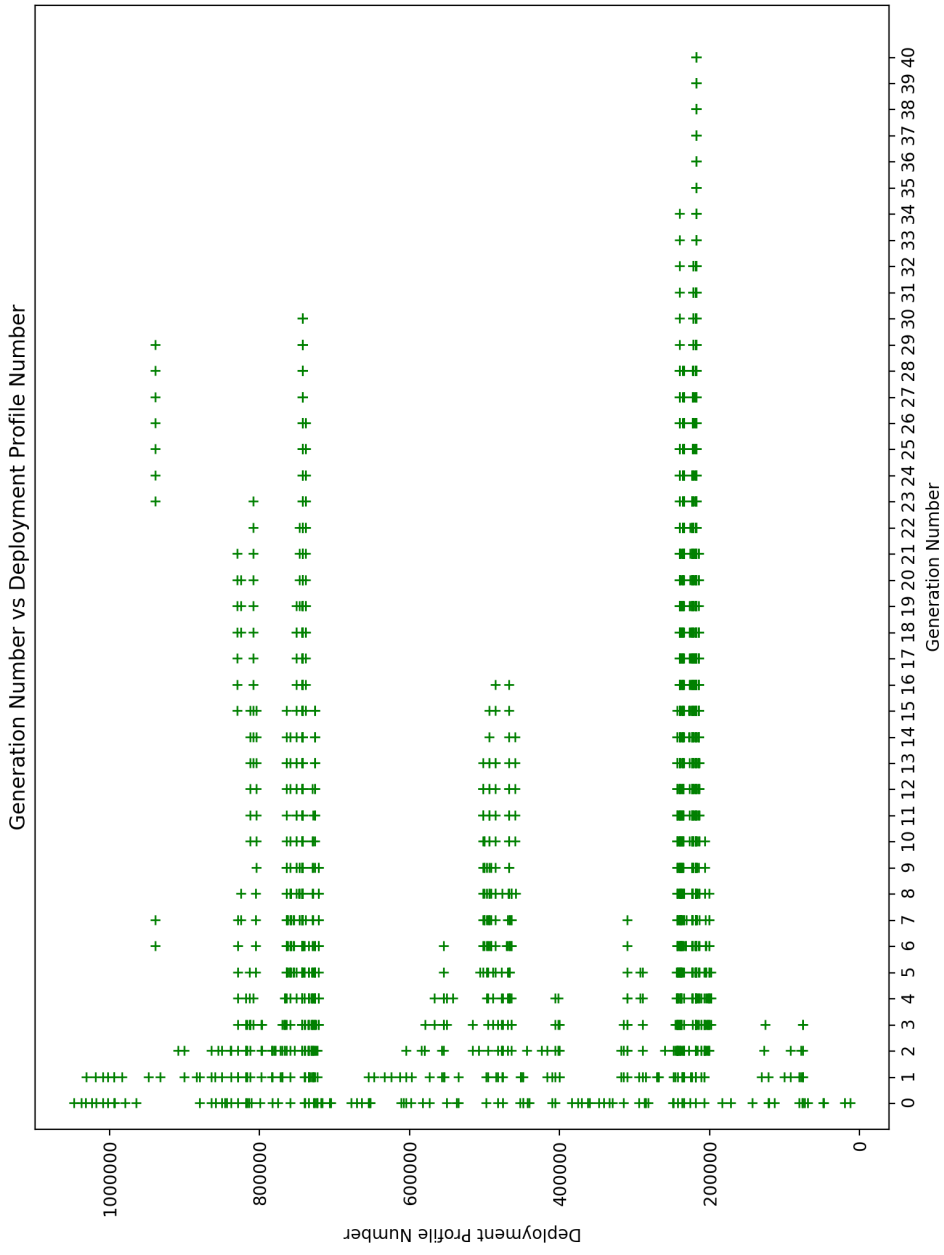

Figure D.19: Deployment Profile Test - Population Size:100 - Number of Generations: 40 - Results #1

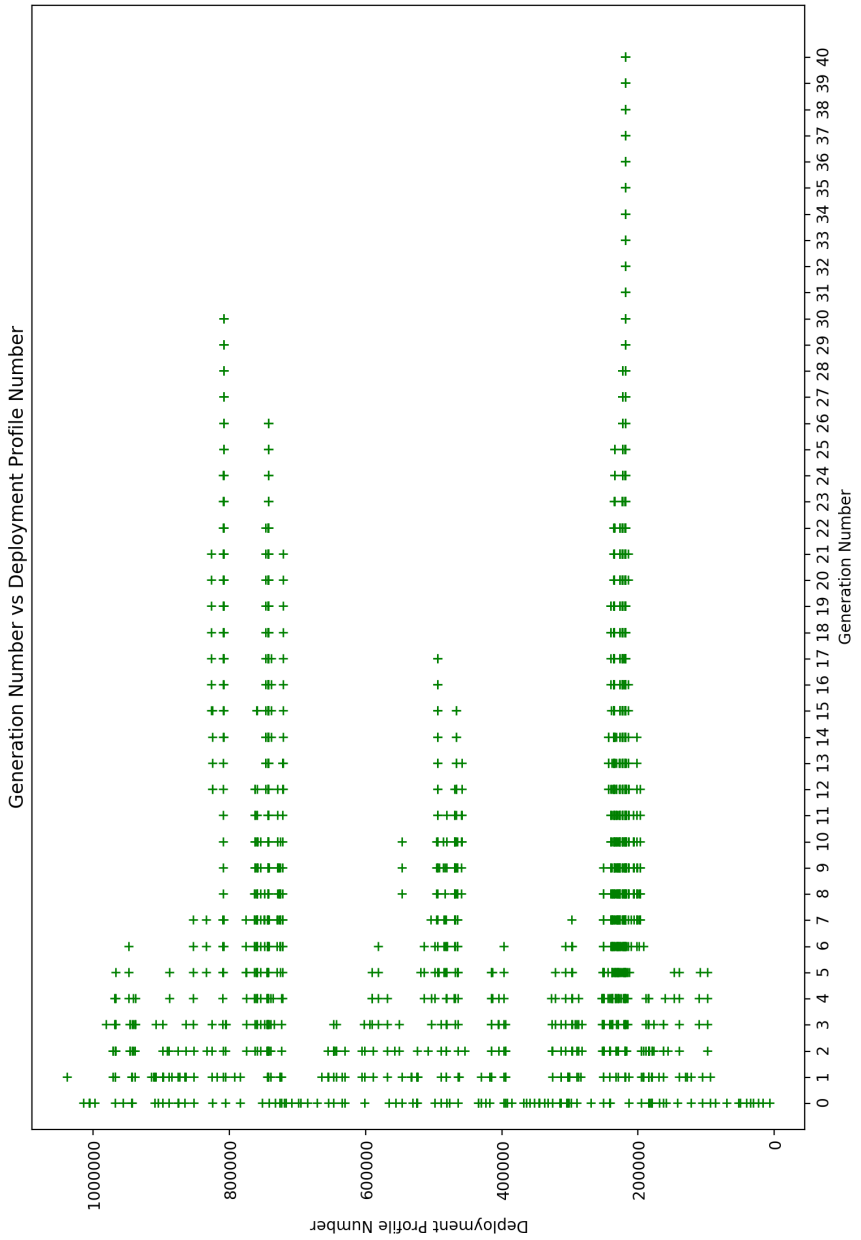

Figure D.20: Deployment Profile Test - Population:100 - Generations:40 - Results #2

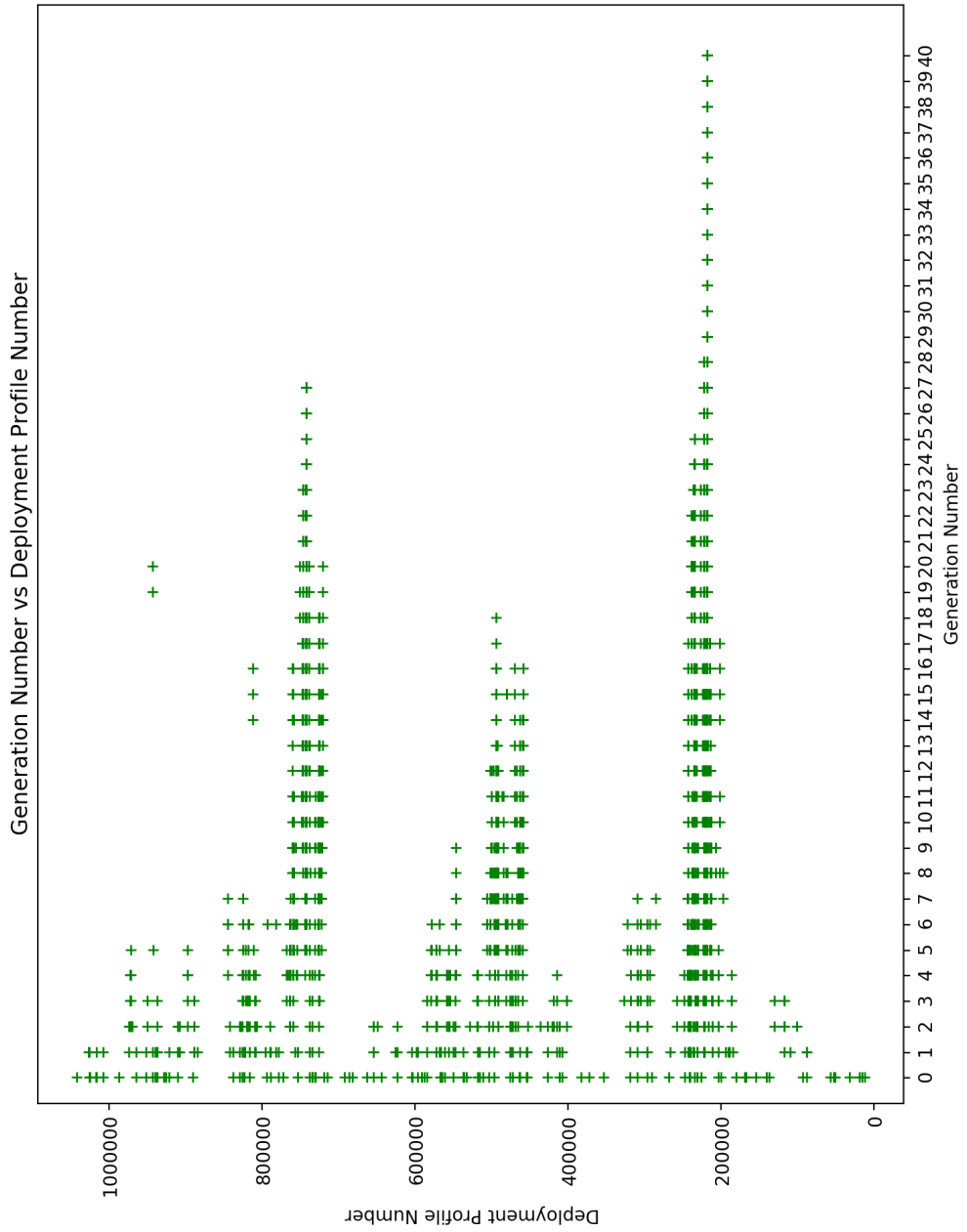

Figure D.21: Deployment Profile Test - Population:100 - Generations:40 - Results #3

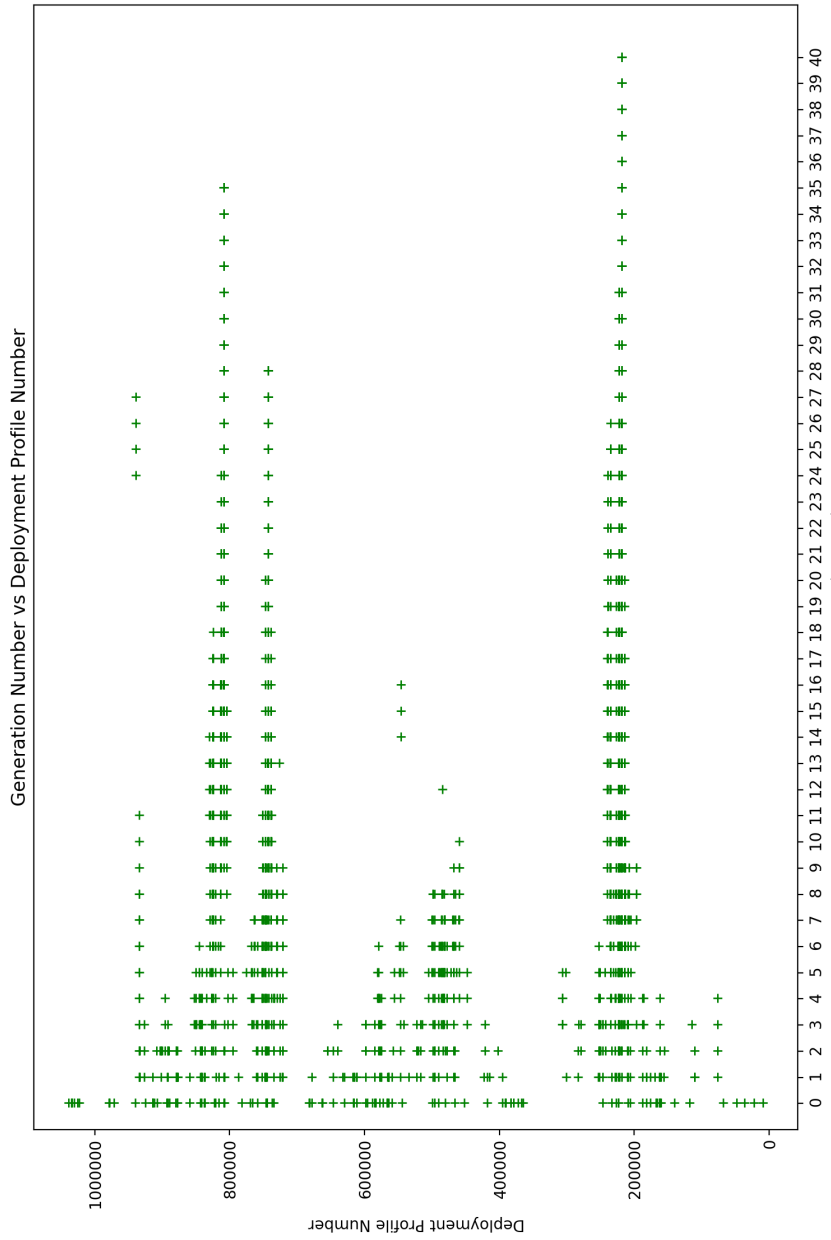

Figure D.22: Deployment Profile Test - Population:100 - Generations:40 - Results #4

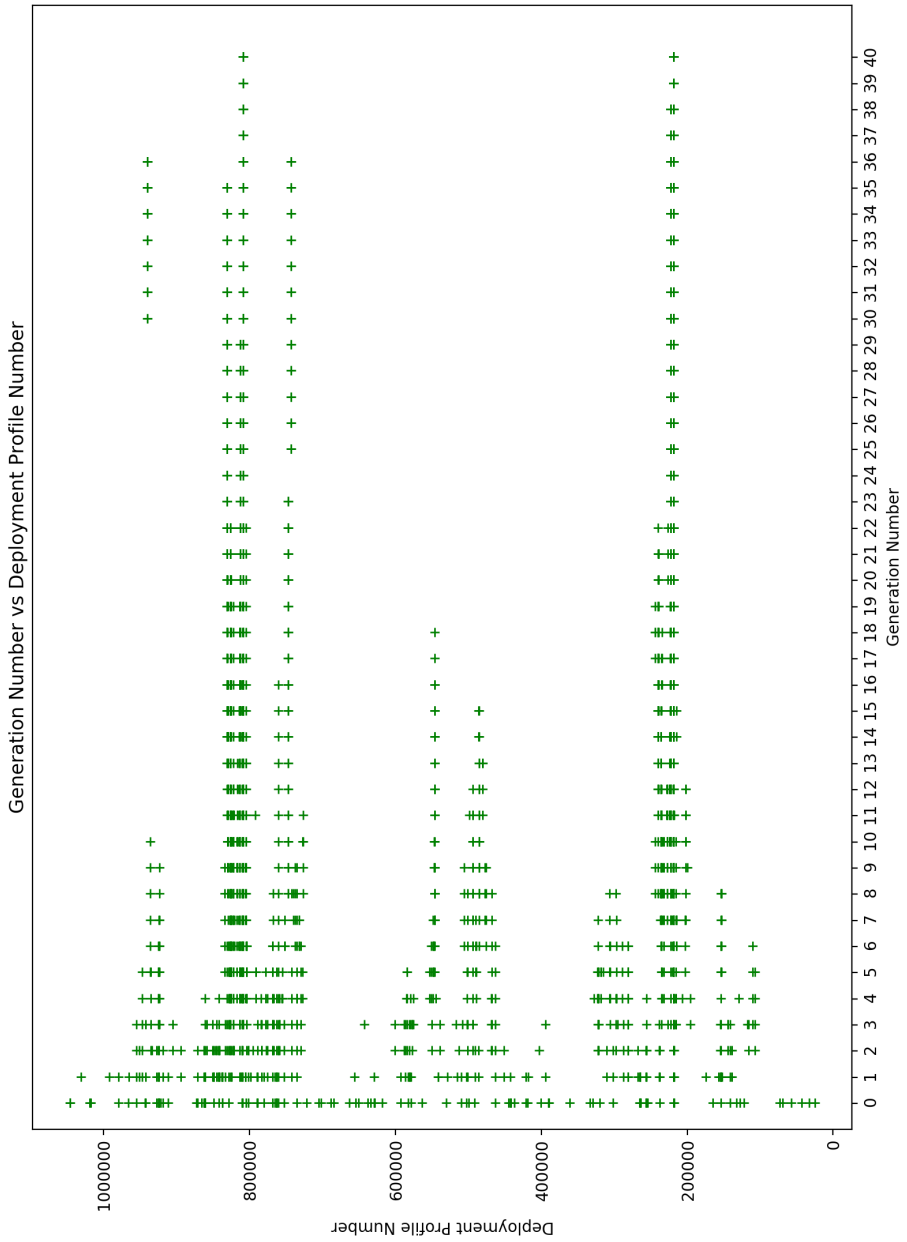

Figure D.23: Deployment Profile Test - Population:100 - Generations:40 - Results #5

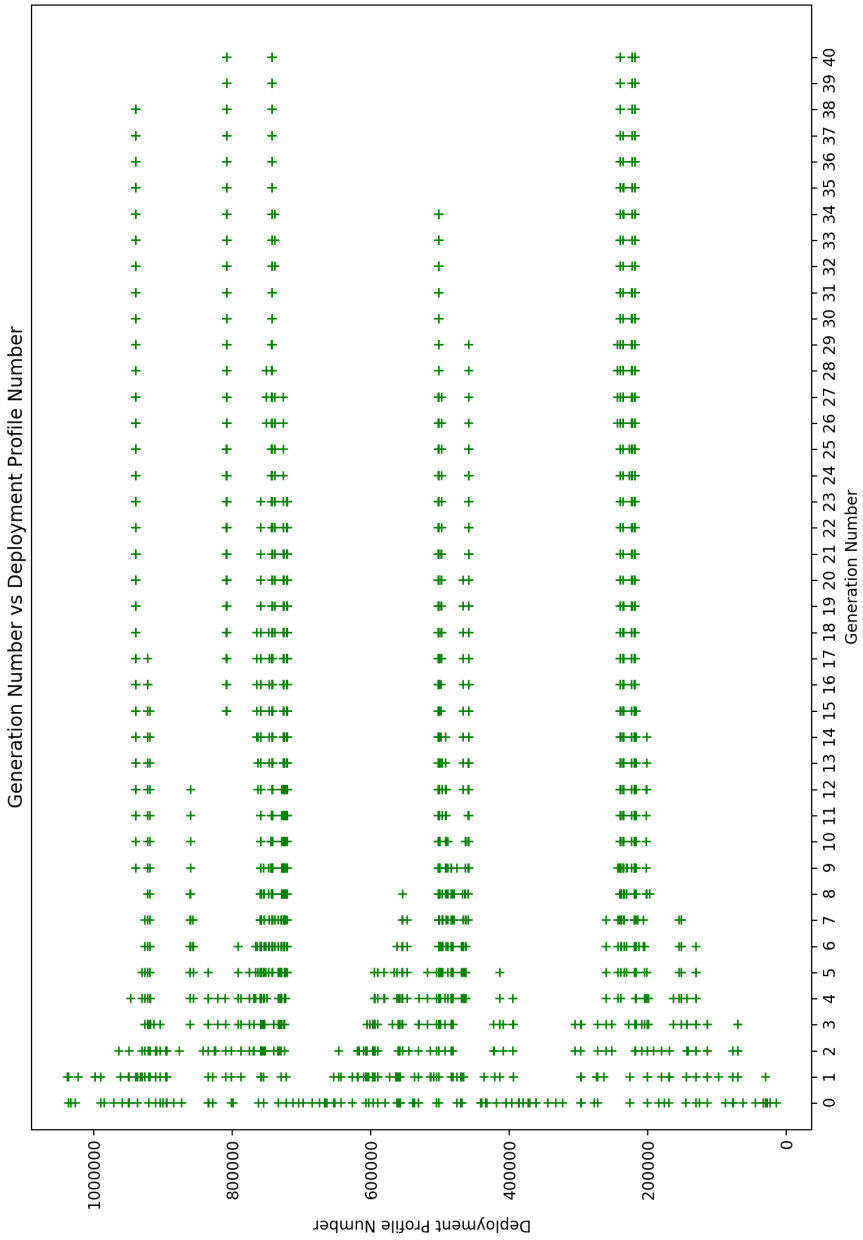

Figure D.24: Deployment Profile Test - Population:100 - Generations:40 - Results #6

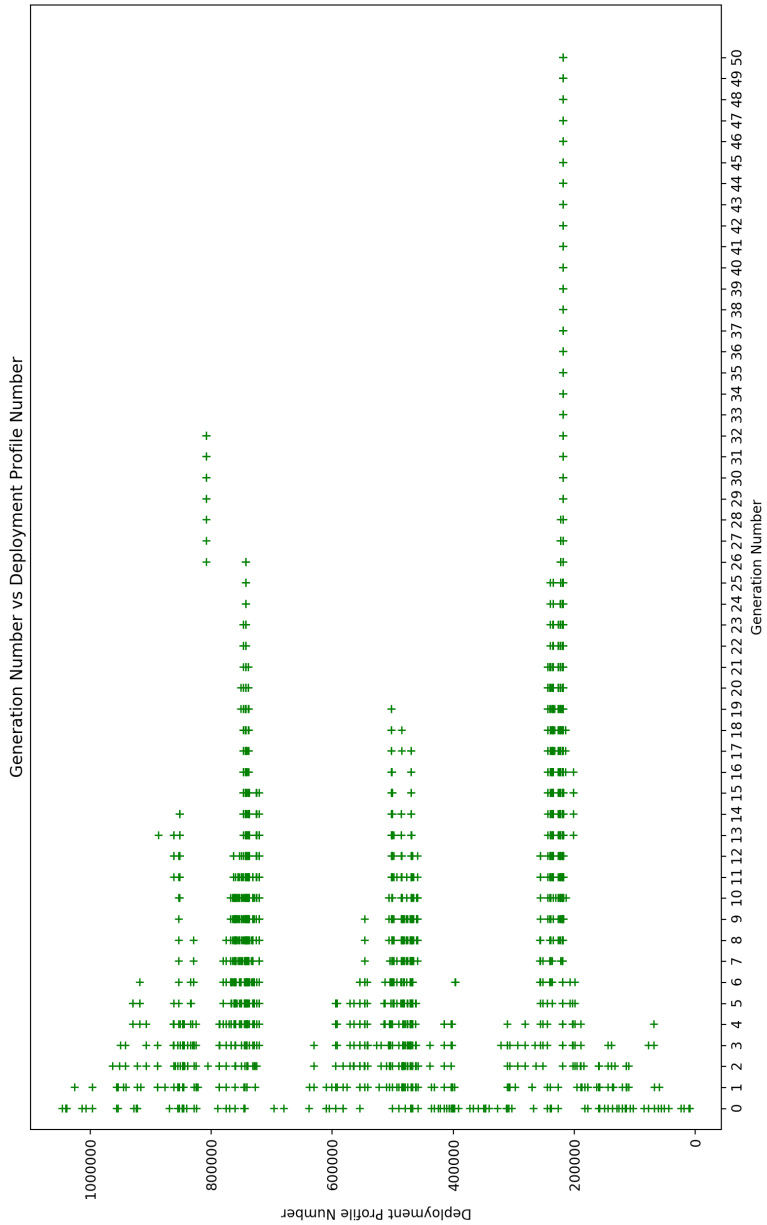

Figure D.25: Deployment Profile Test - Population:100 - Generations: 50 - Results #1

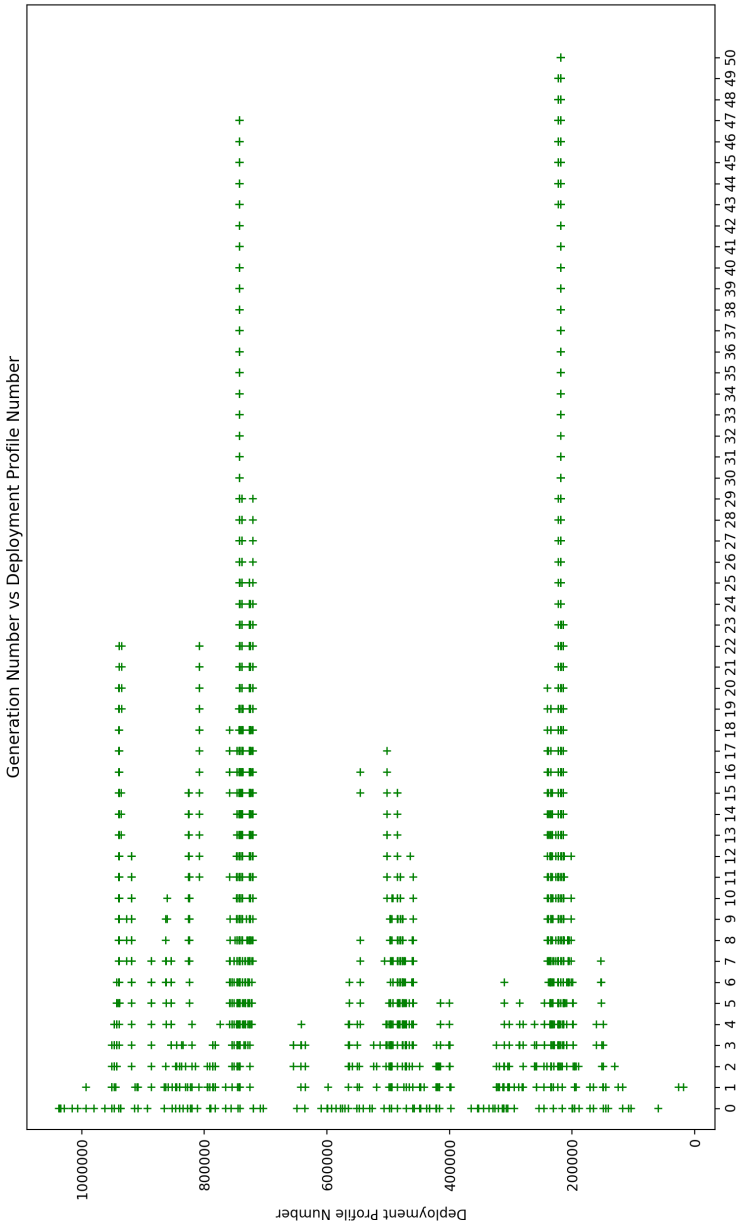

Figure D.26: Deployment Profile Test - Population:100 - Generations:50 - Results #2

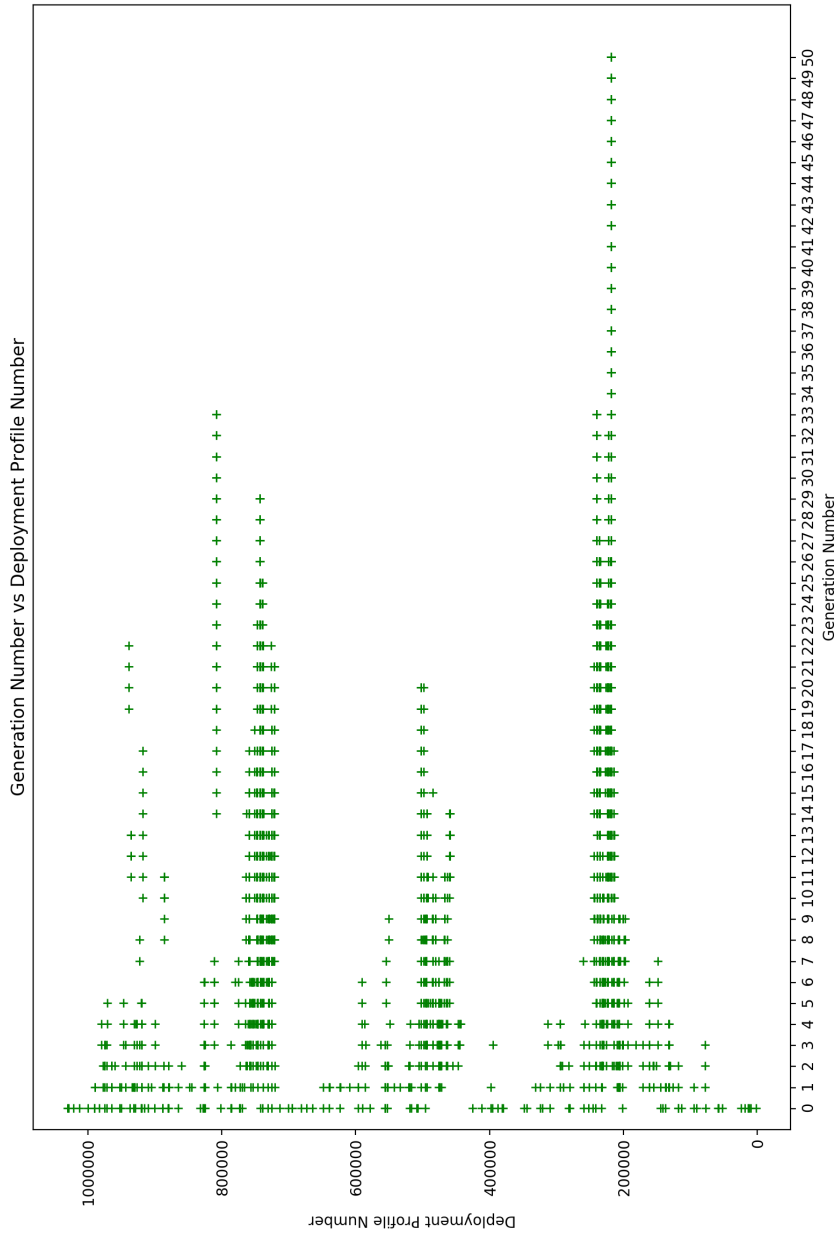

Figure D.27: Deployment Profile Test - Population:100 - Generations: 50 - Results #3

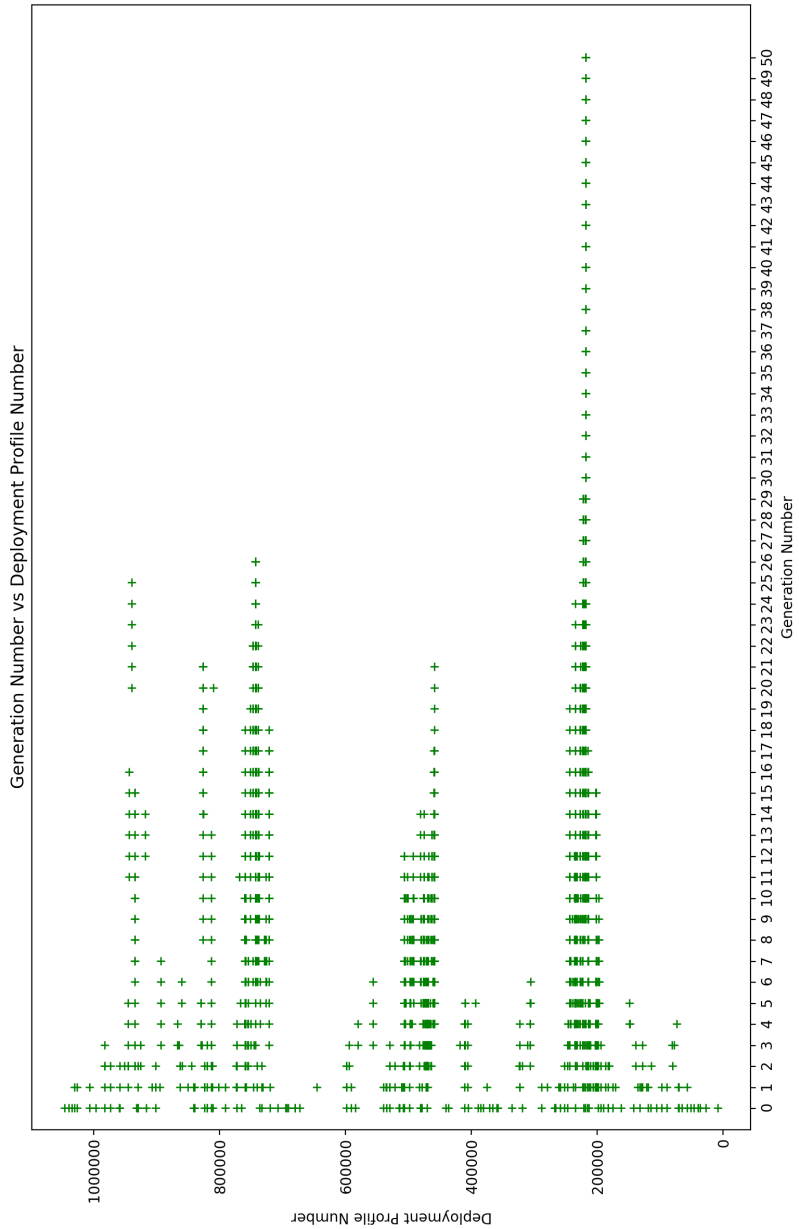

Figure D.28: Deployment Profile Test - Population:100 - Generations:50 - Results #4

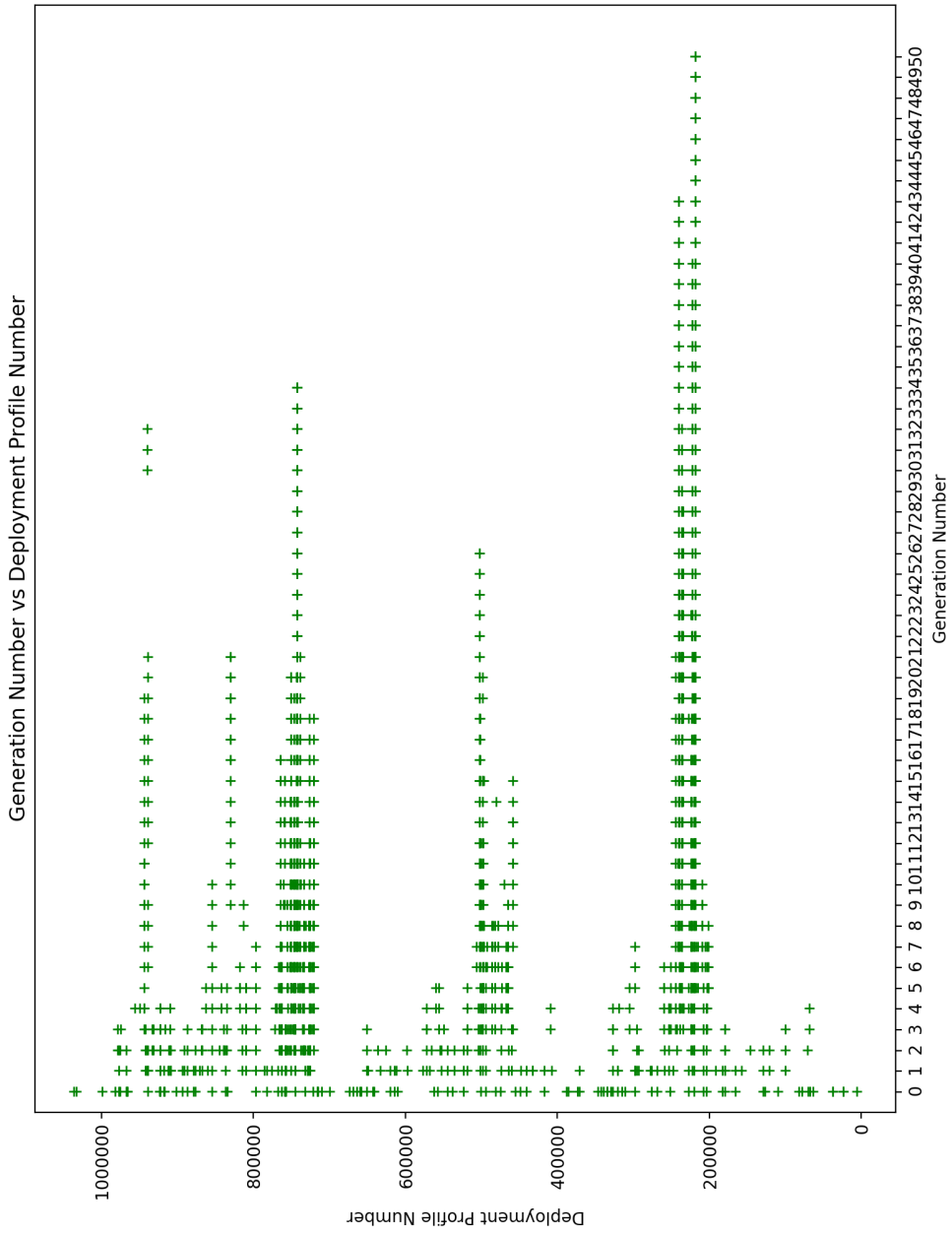

Figure D.29: Deployment Profile Test - Population:100 - Generations:50 - Results #5

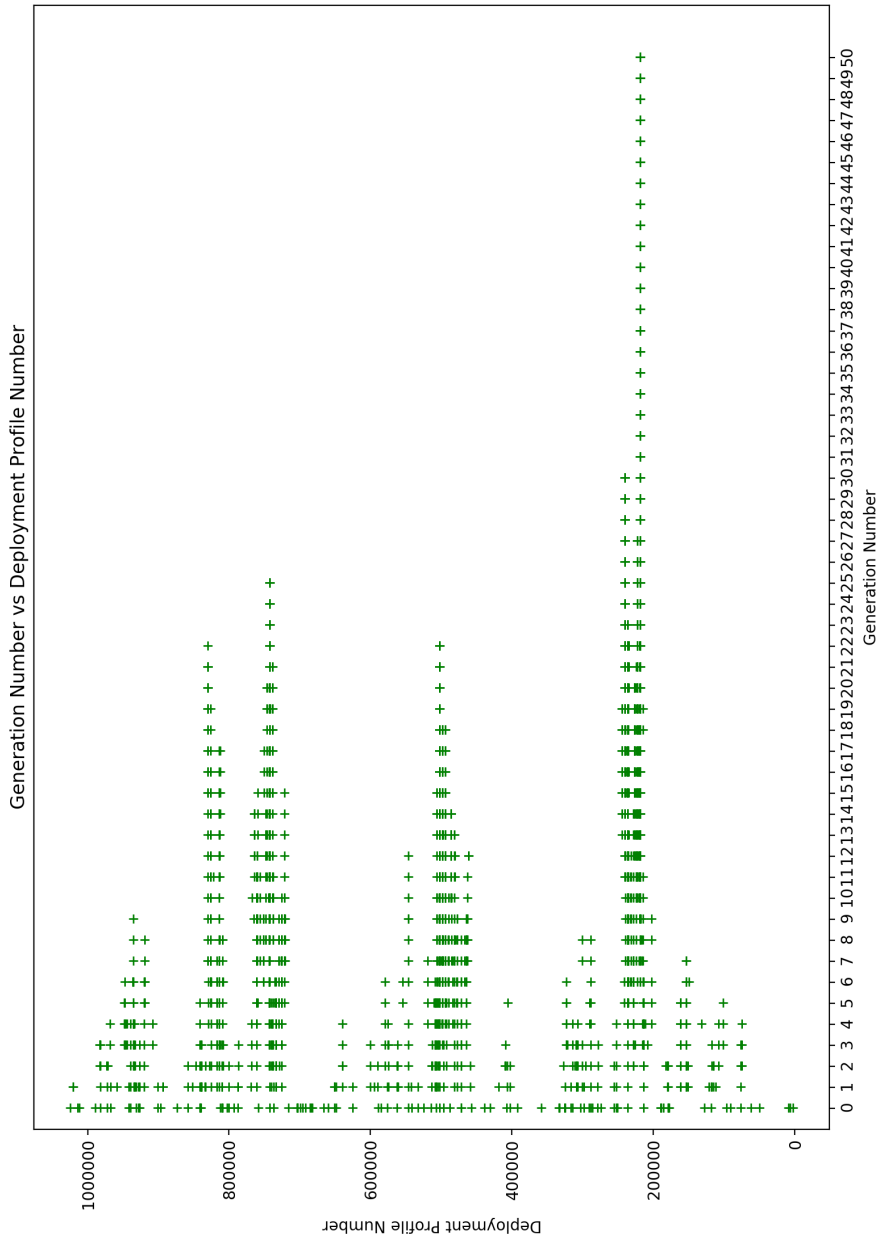

Figure D.30: Deployment Profile Test - Population:100 - Generations:50 - Results #6

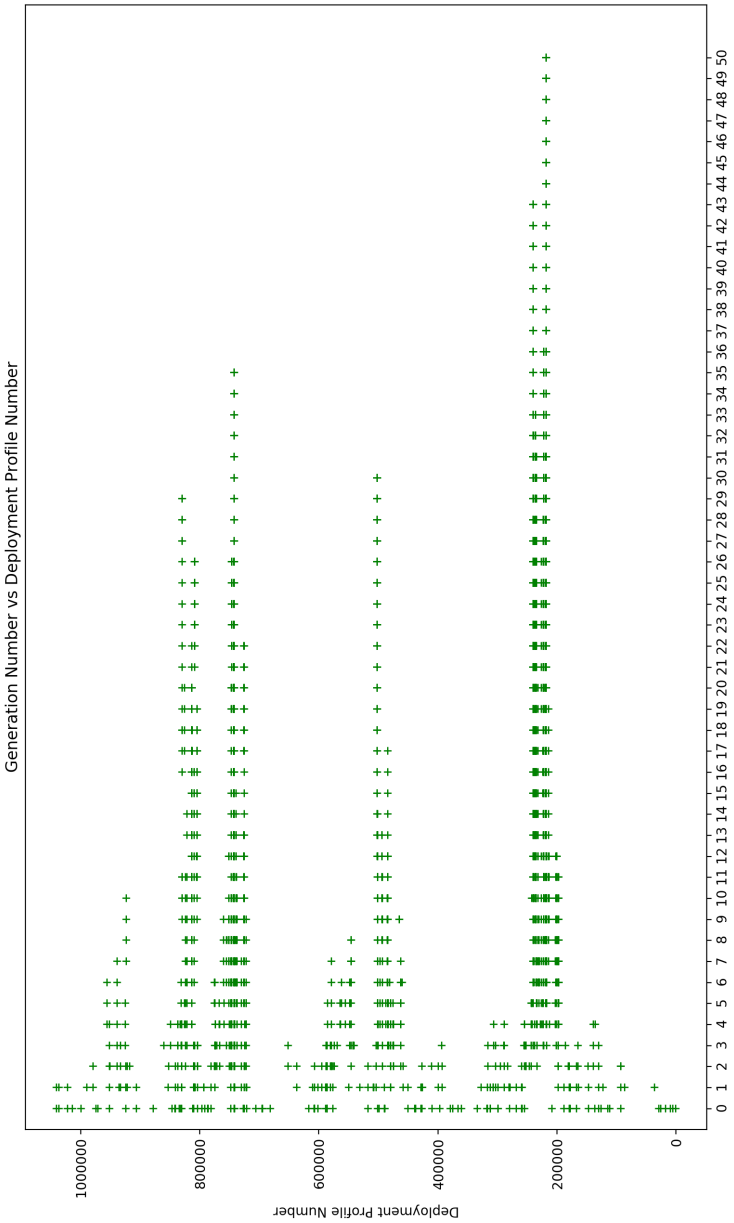

Figure D.31: Deployment Profile Test - Population:90 - Generations:50 - Results #1

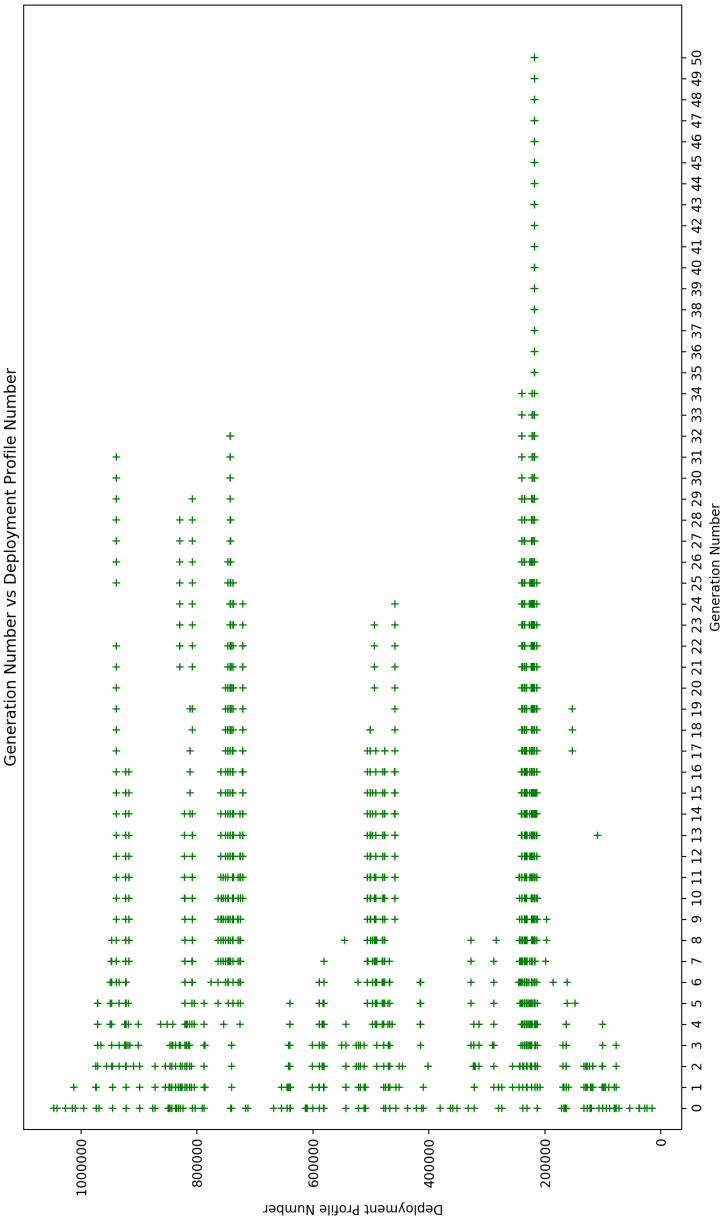

Figure D.32: Deployment Profile Test - Population:90 - Generations:50 - Results #2

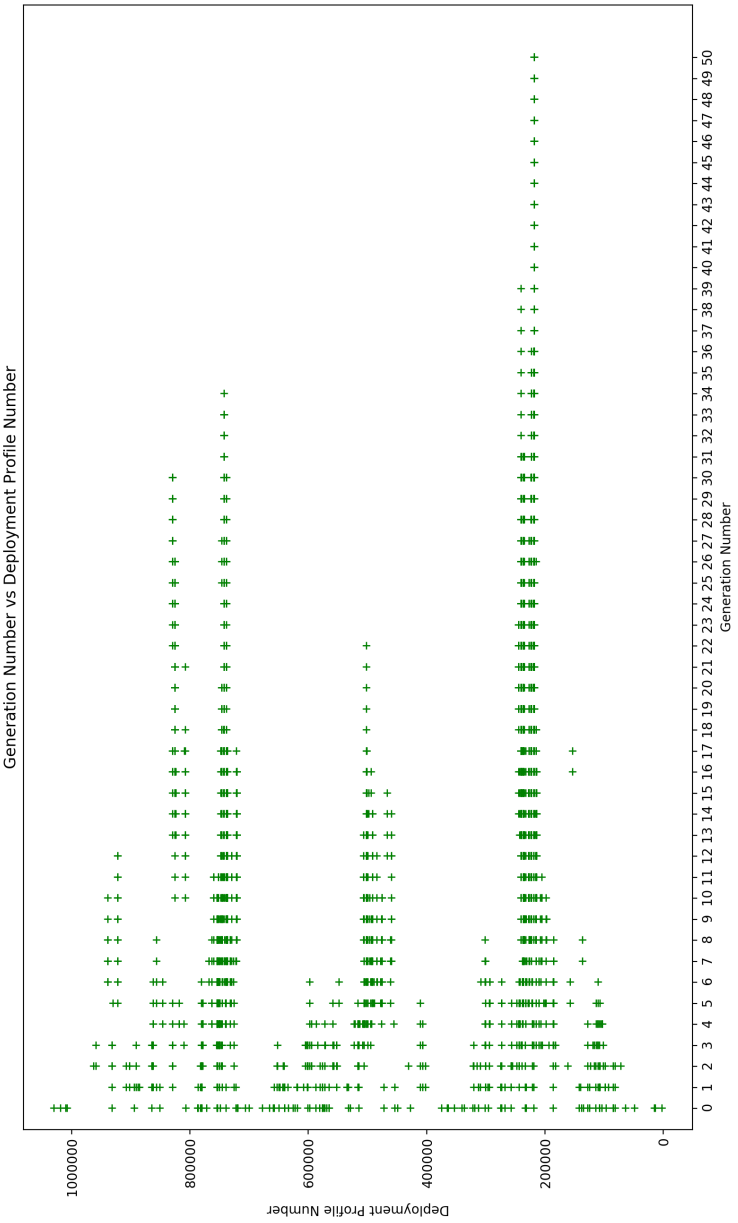

Figure D.33: Deployment Profile Test - Population:90 - Generations:50 - Results #3

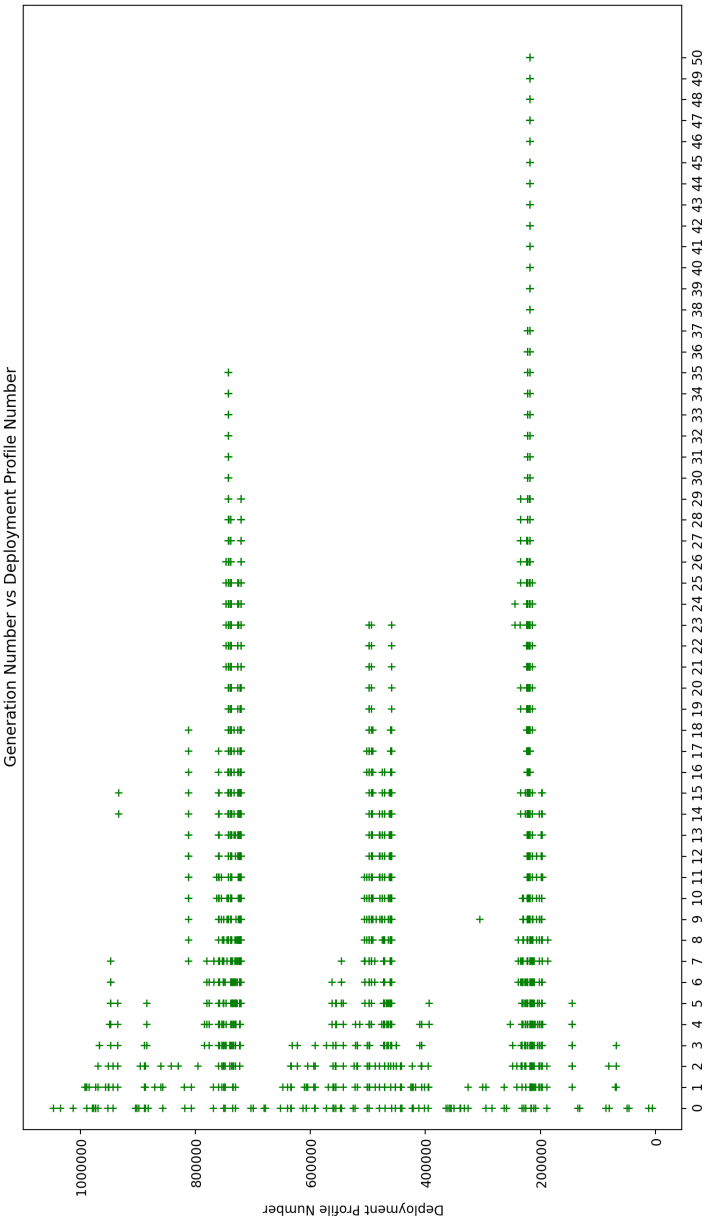

Figure D.34: Deployment Profile Test - Population:90 - Generations:50 - Results #4

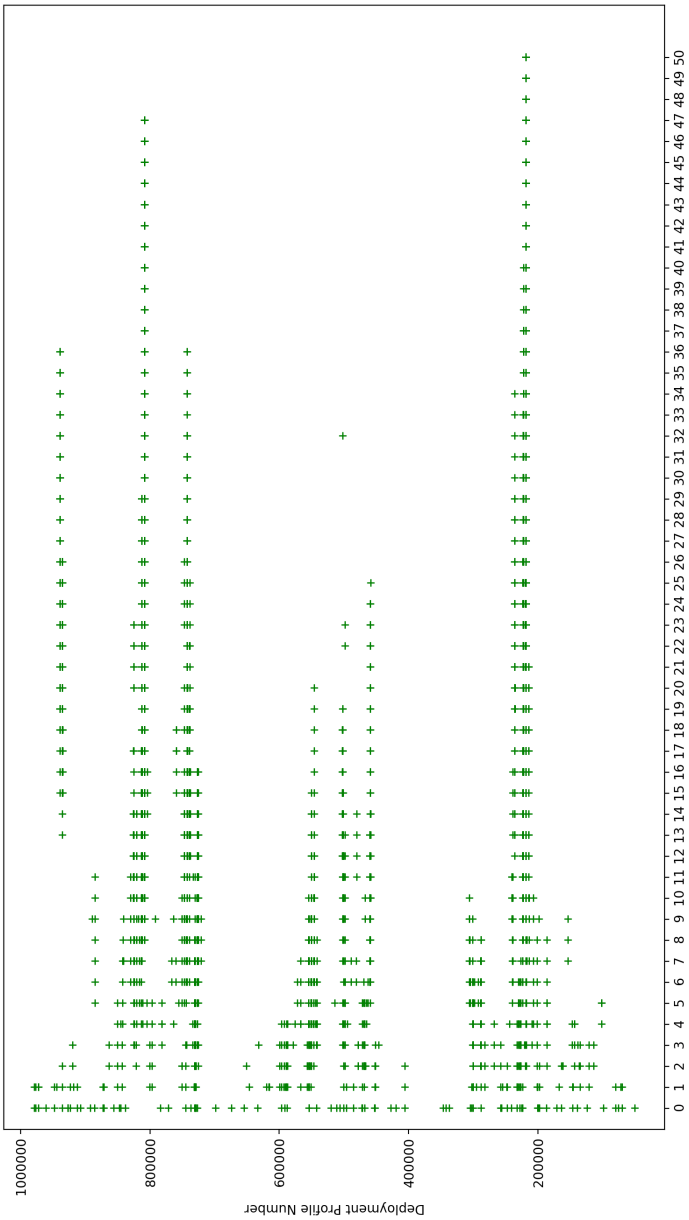

Figure D.35: Deployment Profile Test - Population:80 - Generations:50 - Results #1

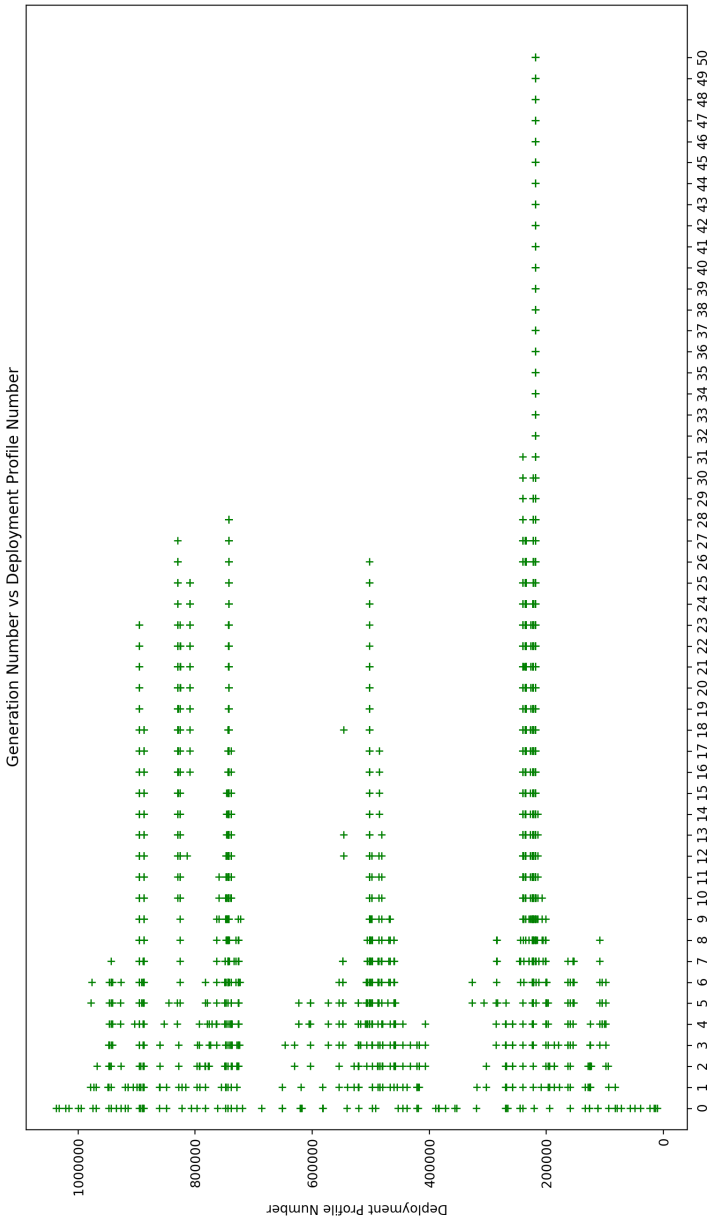

Figure D.36: Deployment Profile Test - Population:80 - Generations:50 - Results #2

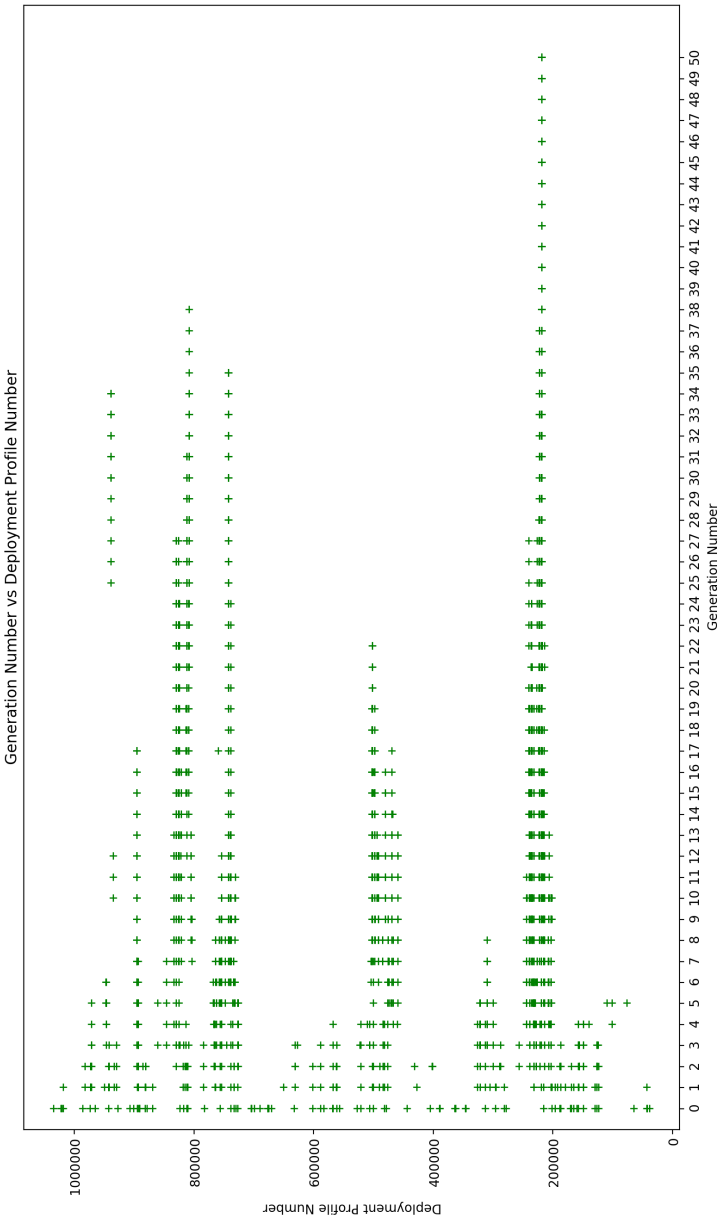

Figure D.37: Deployment Profile Test - Population:80 - Generations:50 - Results #3

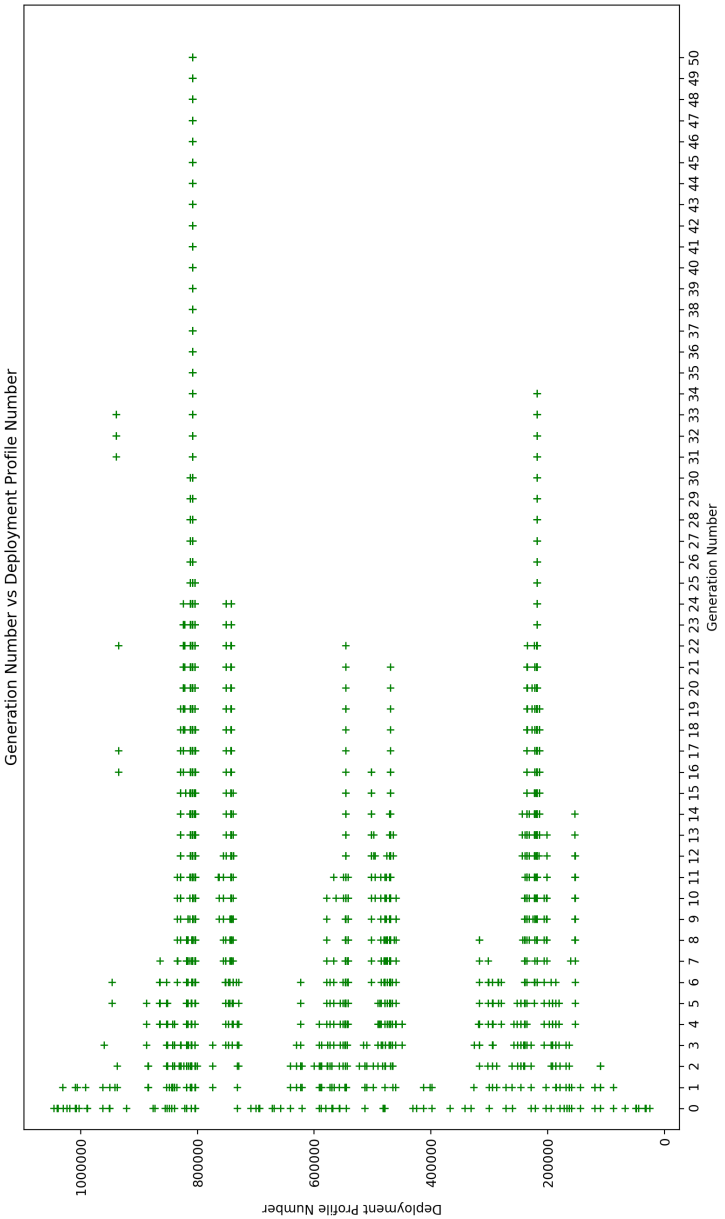

Figure D.38: Deployment Profile Test - Population:80 - Generations:50 - Results #4

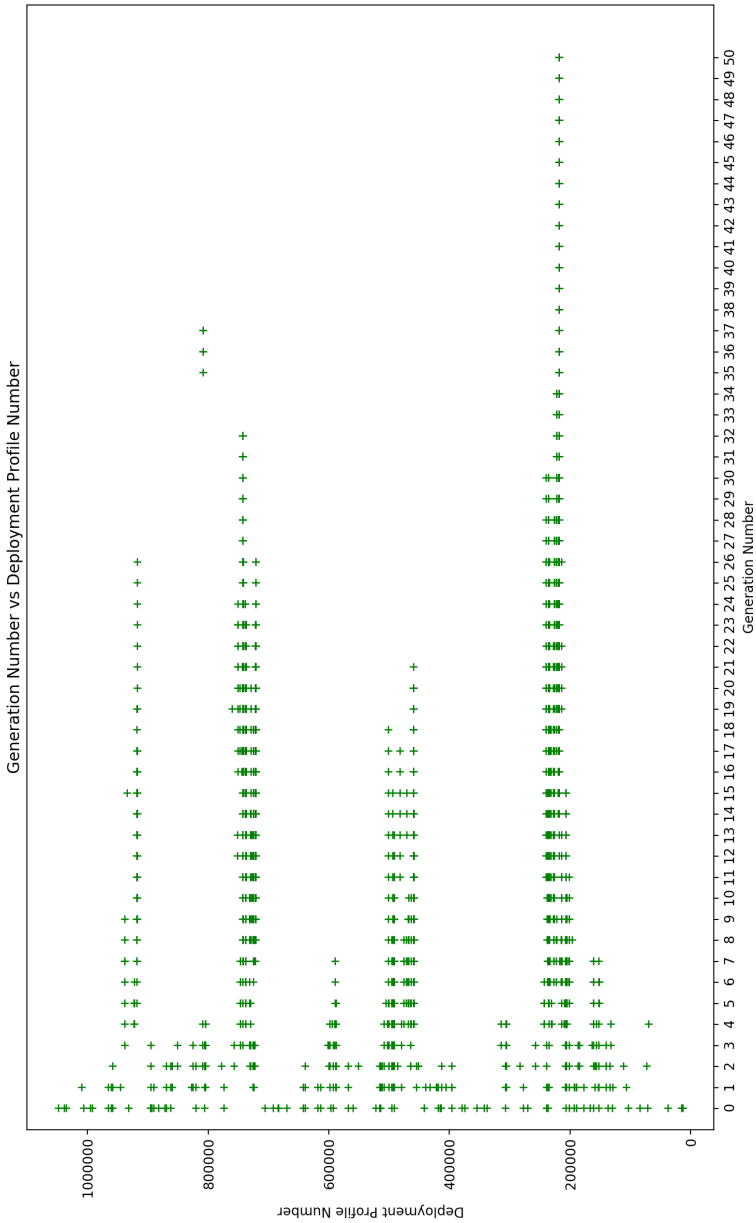

Figure D.39: Deployment Profile Test - Population:70 - Generations:50 - Results #1

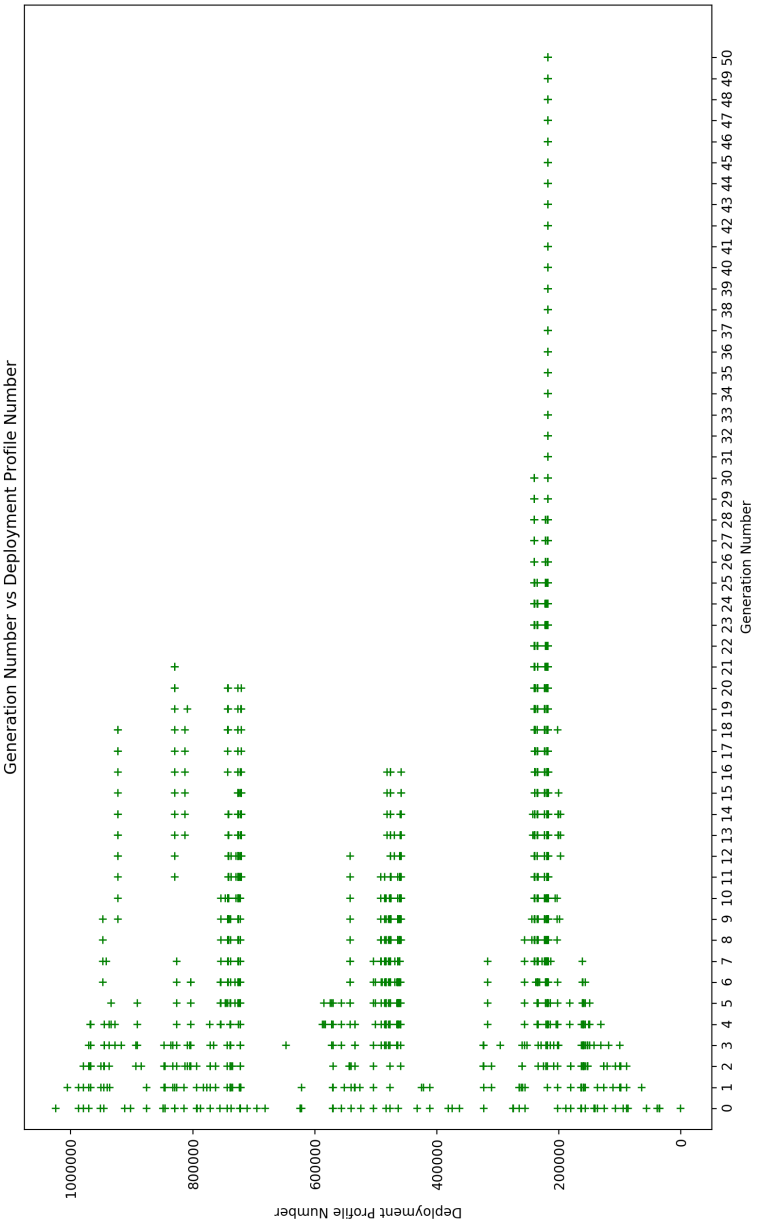

Figure D.40: Deployment Profile Test - Population:70 - Generations:50 - Results #2

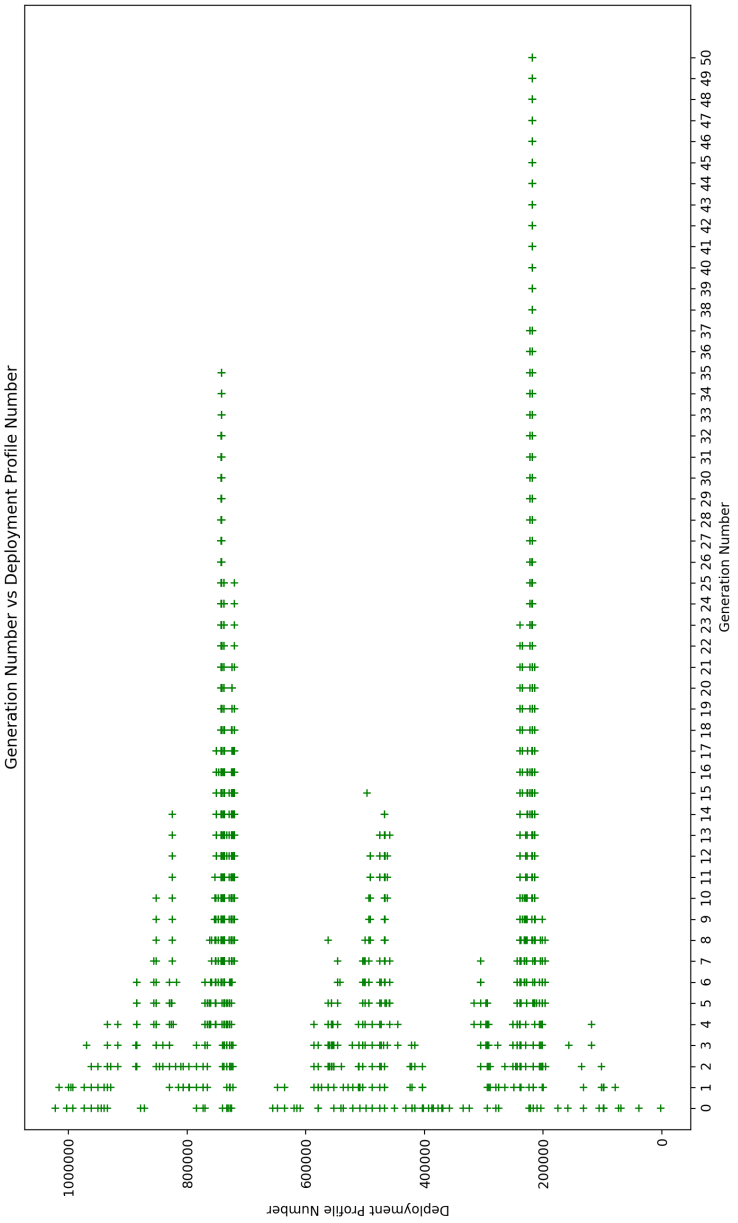

Figure D.41: Deployment Profile Test - Population:70 - Generations:50 - Results #3

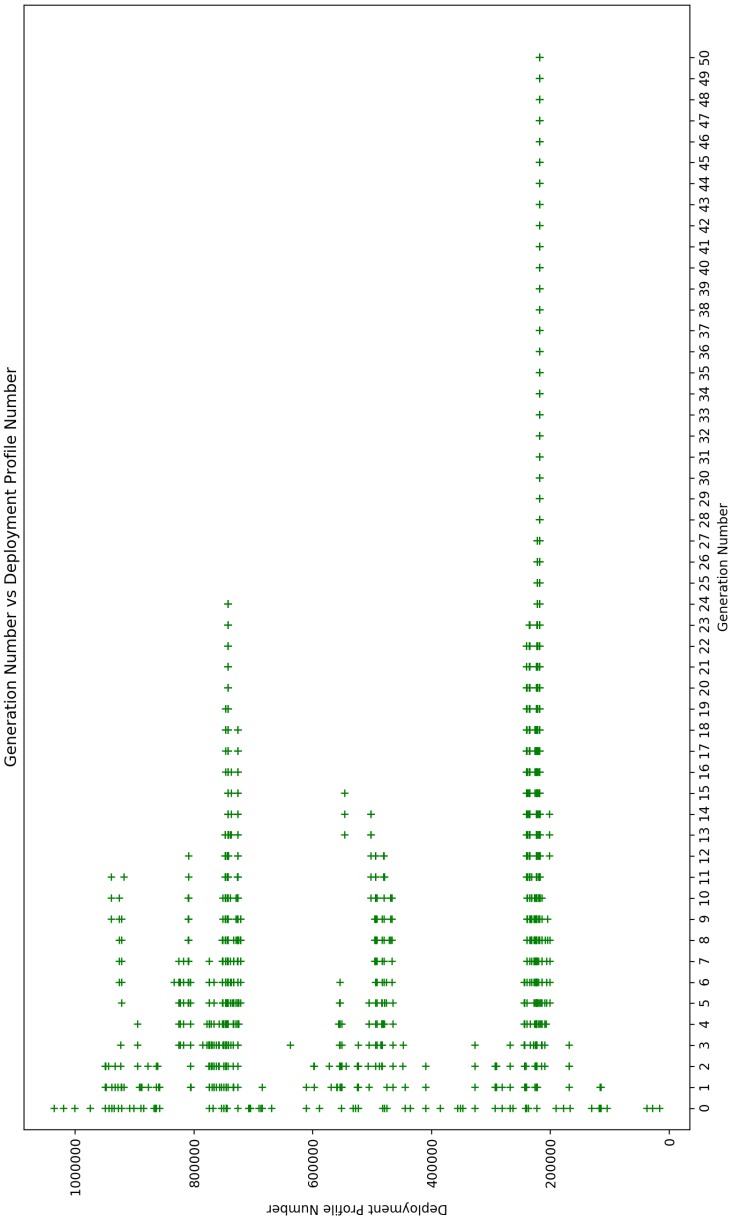

Figure D.42: Deployment Profile Test - Population:70 - Generations:50 - Results #4

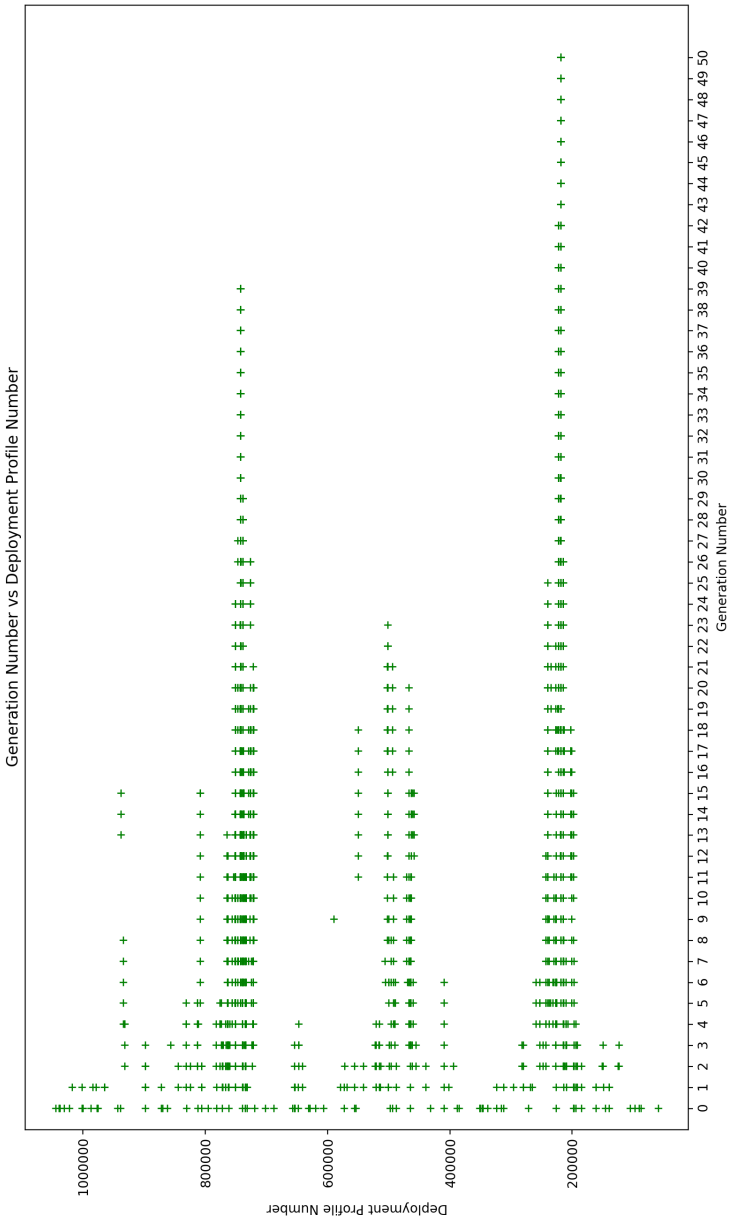

Figure D.43: Deployment Profile Test - Population:70 - Generations:50 - Results #5

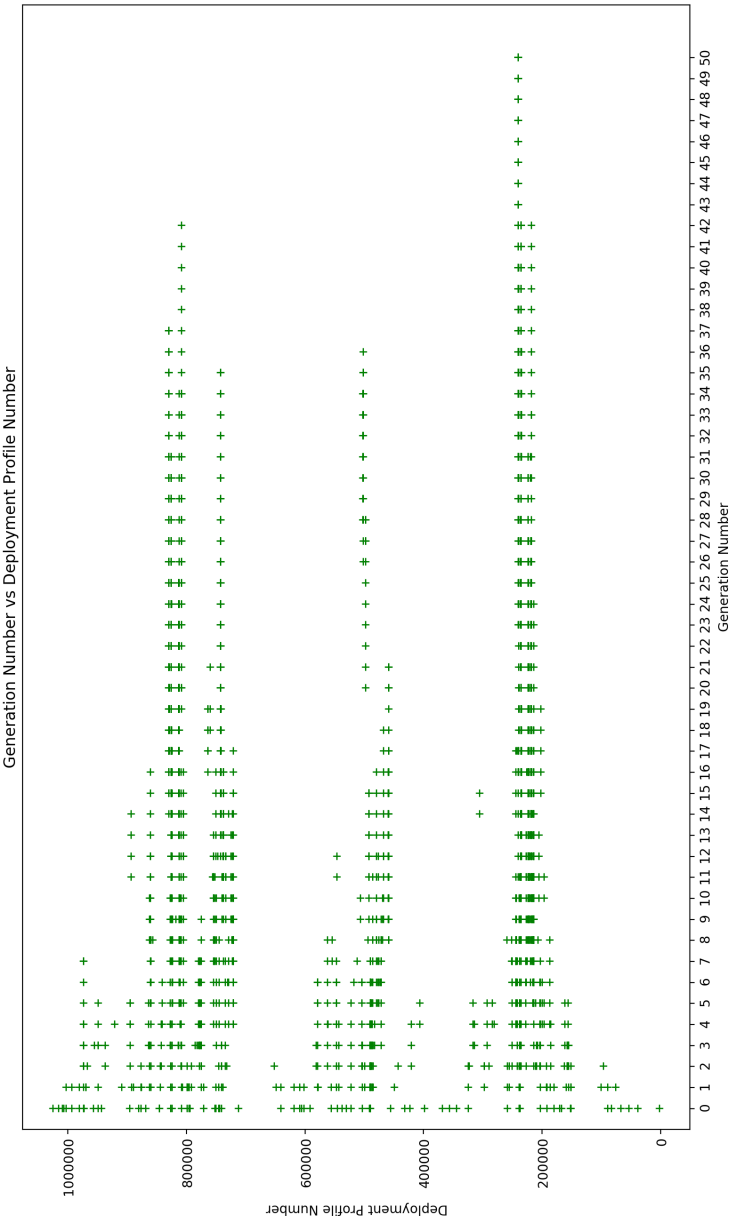

Figure D.44: Deployment Profile Test - Population:70 - Generations:50 - Results #6

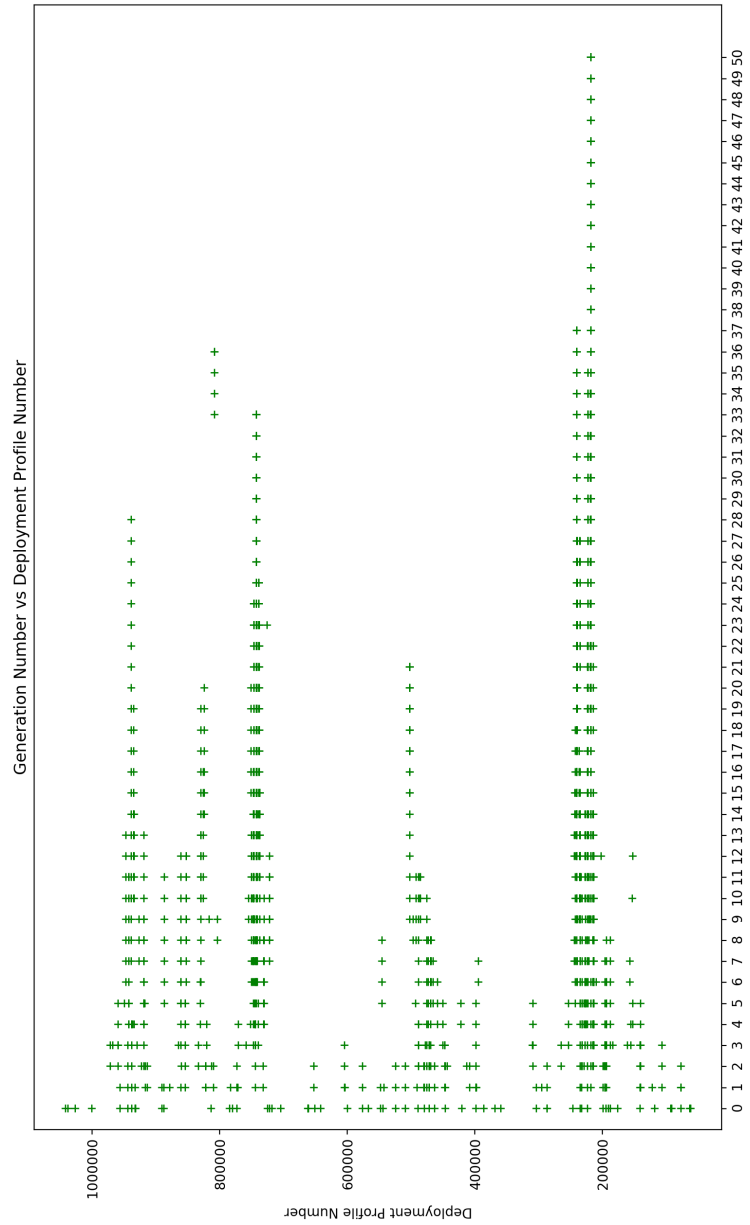

Figure D.45: Deployment Profile Test - Population:60 - Generations:50 - Results #1

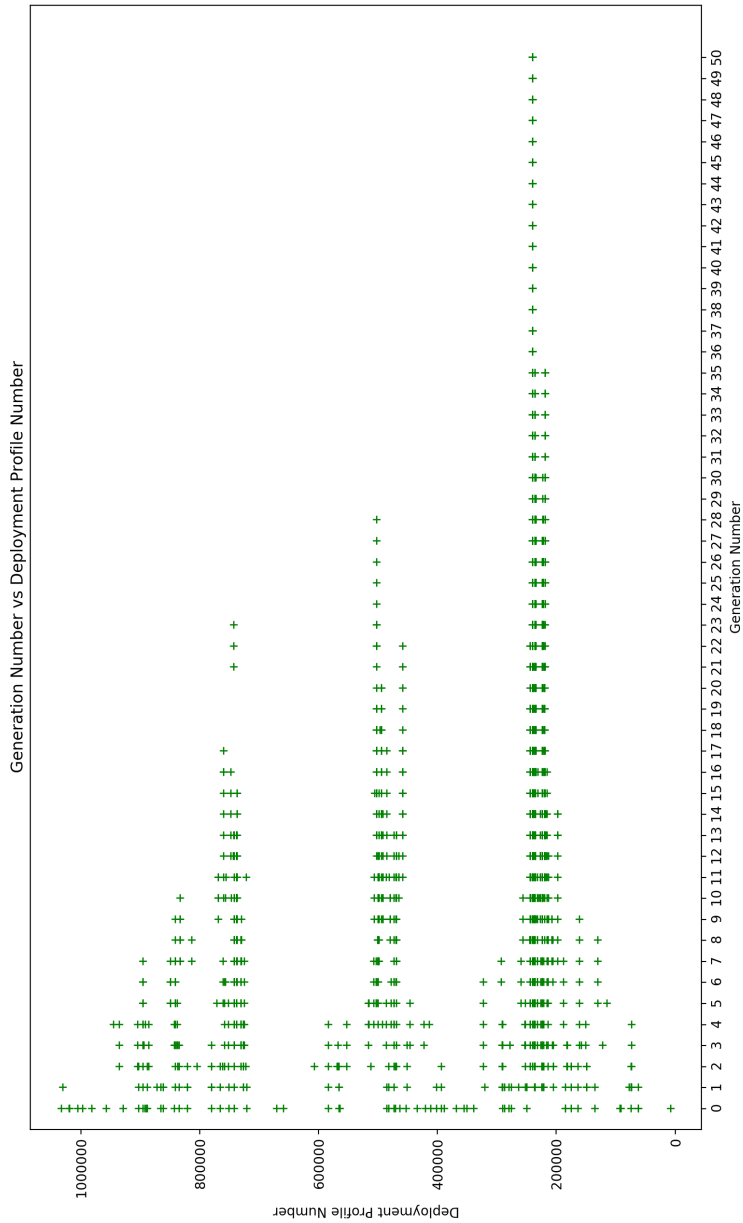

Figure D.46: Deployment Profile Test - Population:60 - Generations:50 - Results #2

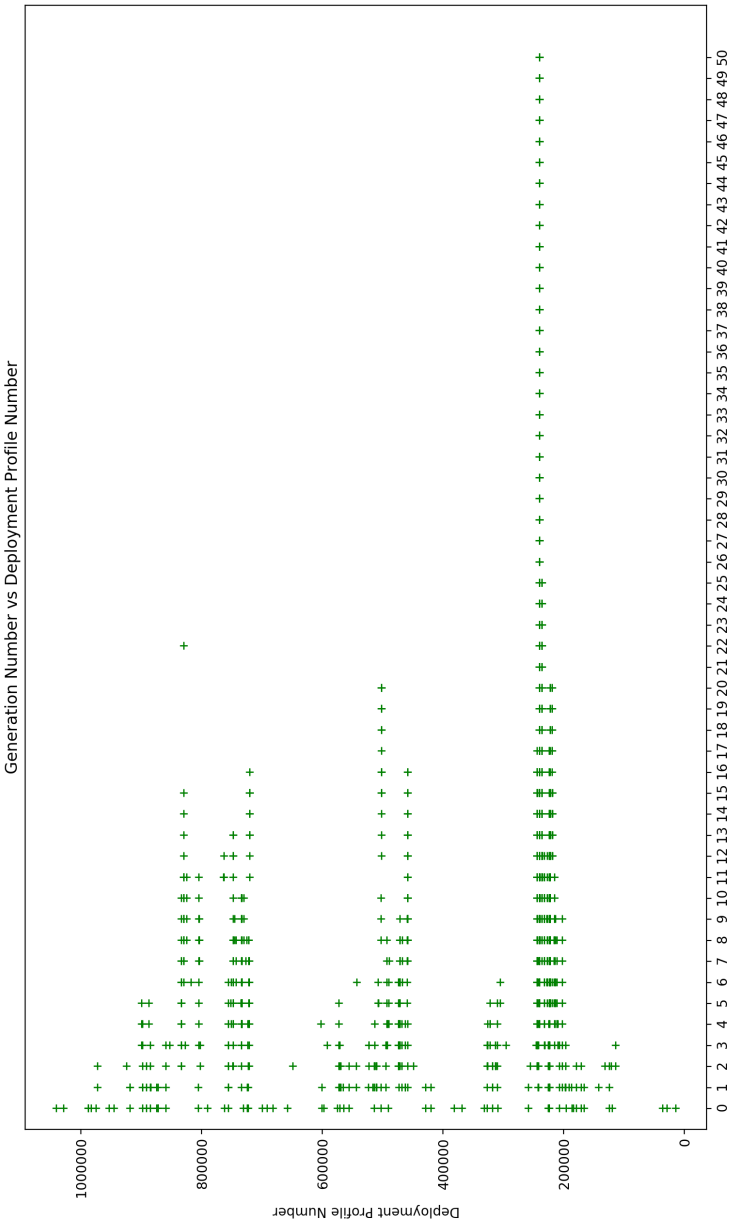

Figure D.47: Deployment Profile Test - Population:60 - Generations:50 - Results #3

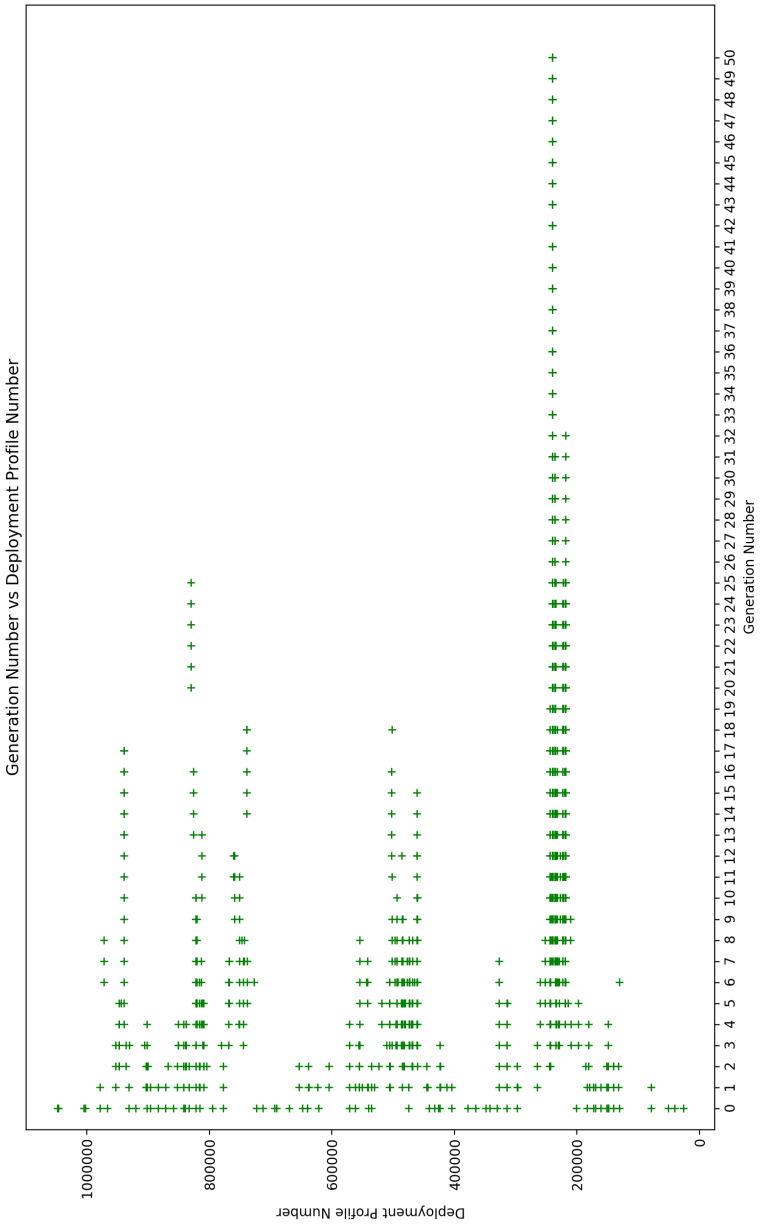

Figure D.48: Deployment Profile Test - Population:60 - Generations:50 - Results #4

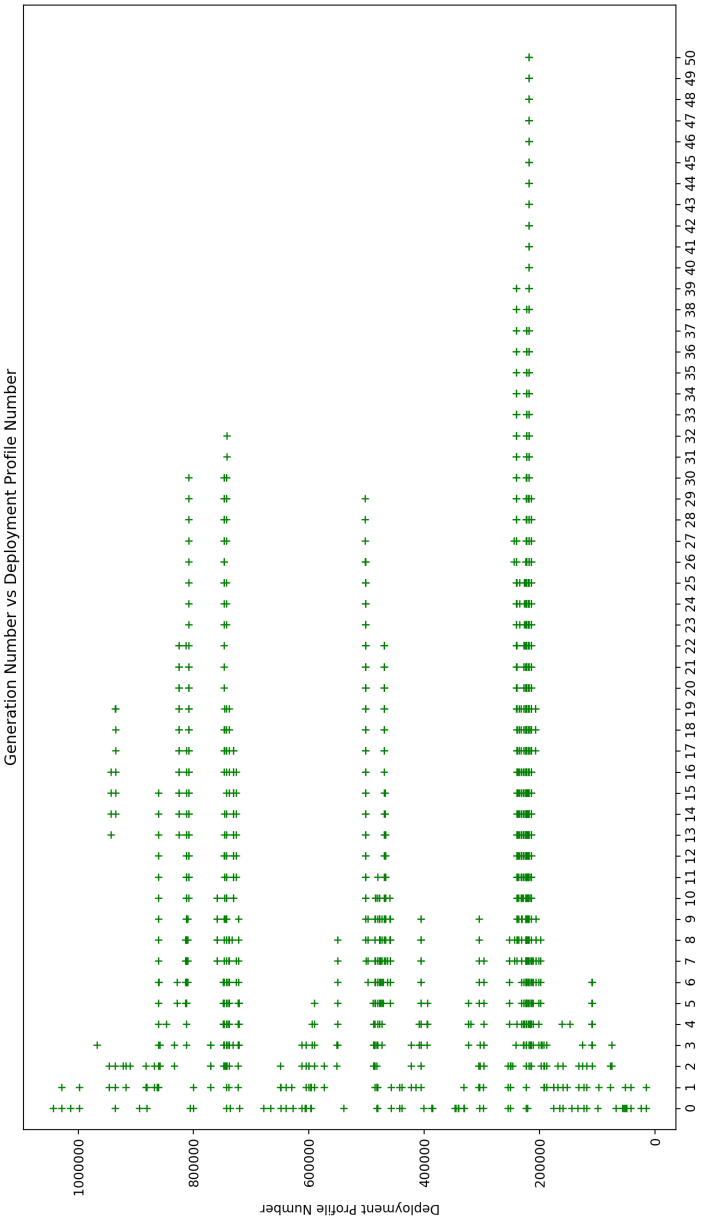

Figure D.49: Deployment Profile Test - Population:60 - Generations:50 - Results #5

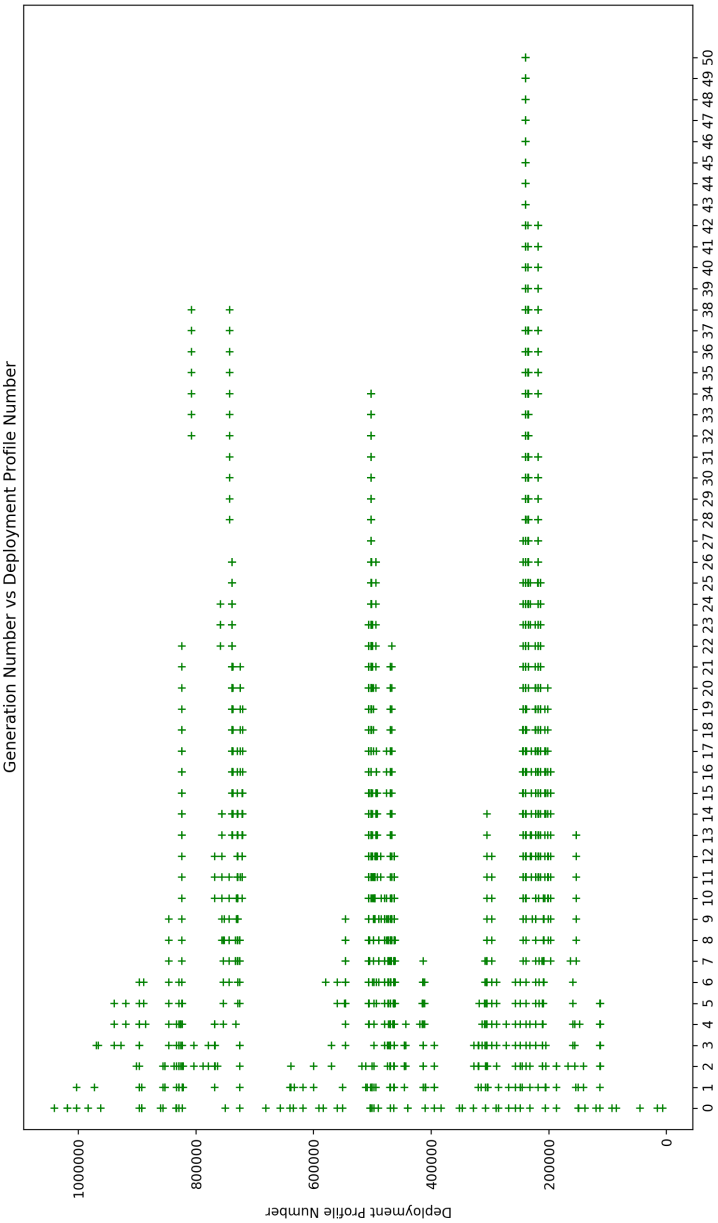

Figure D.50: Deployment Profile Test - Population:60 - Generations:50 - Results #6

D.6 Predicted and Measured Critical Chain Convergence

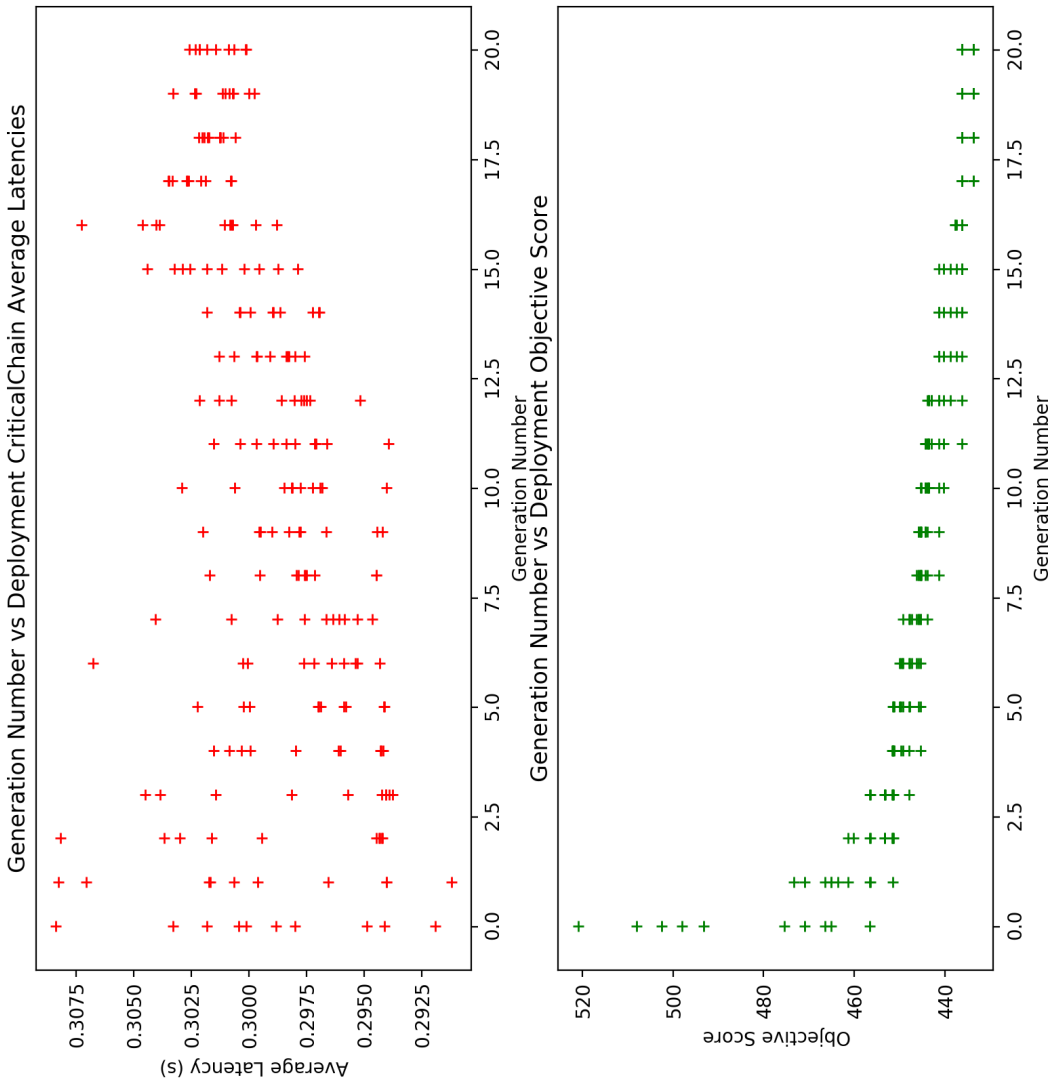

Figure D.51: 16Gb Network Configuration

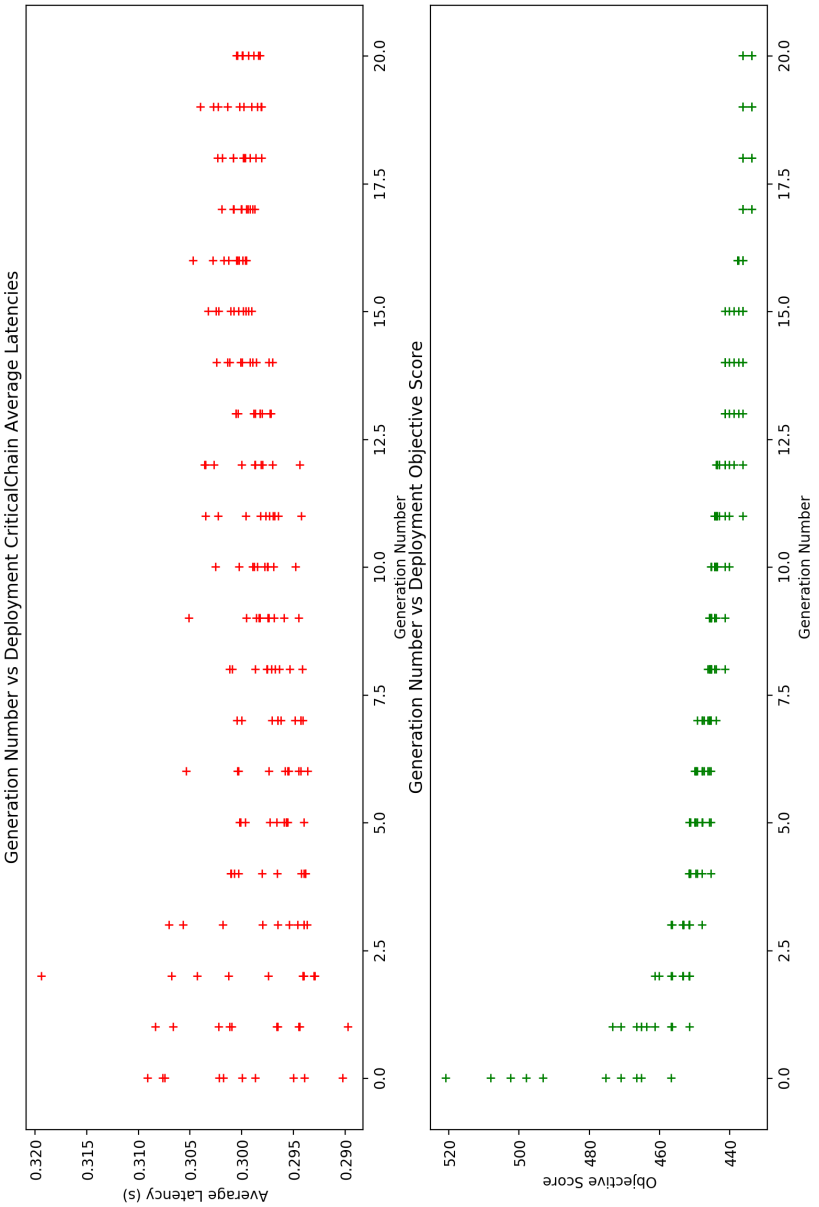

Figure D.52: 16Gb Network Configuration

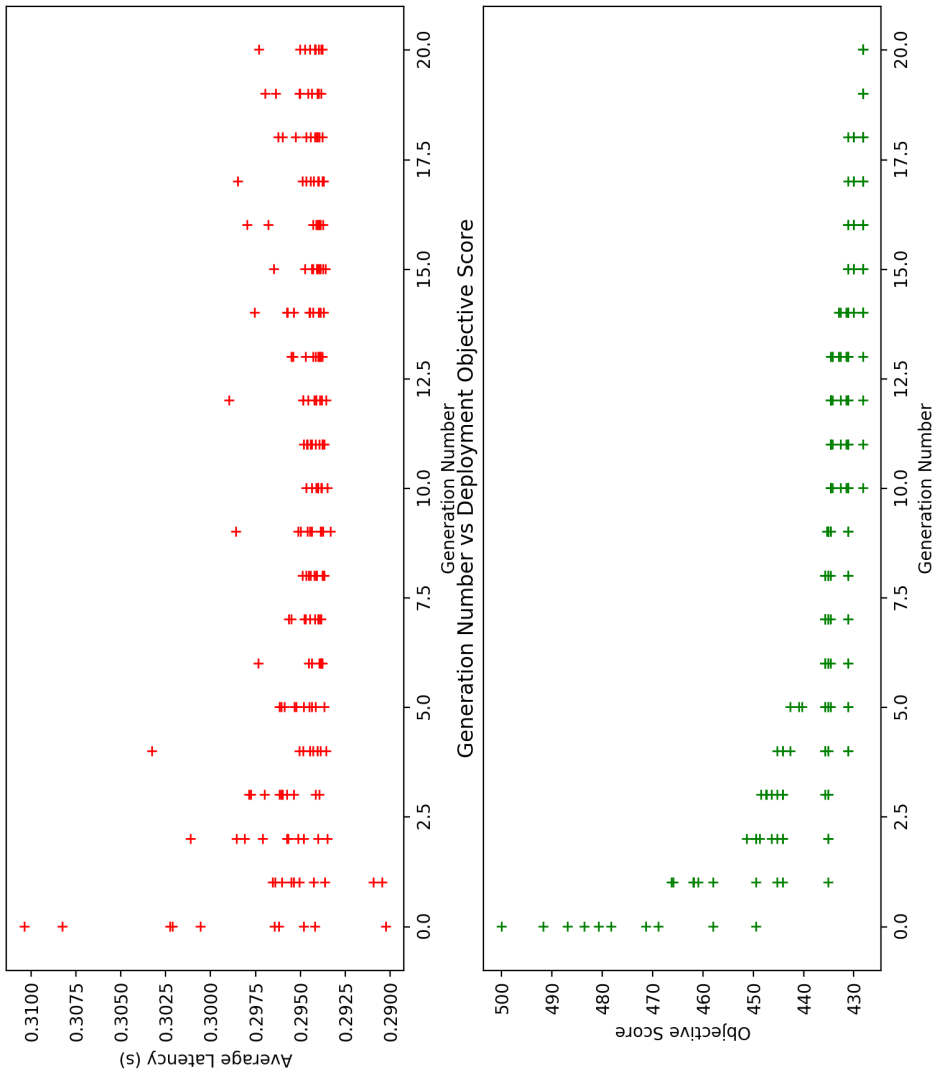

Figure D.53: 16Gb Network Configuration

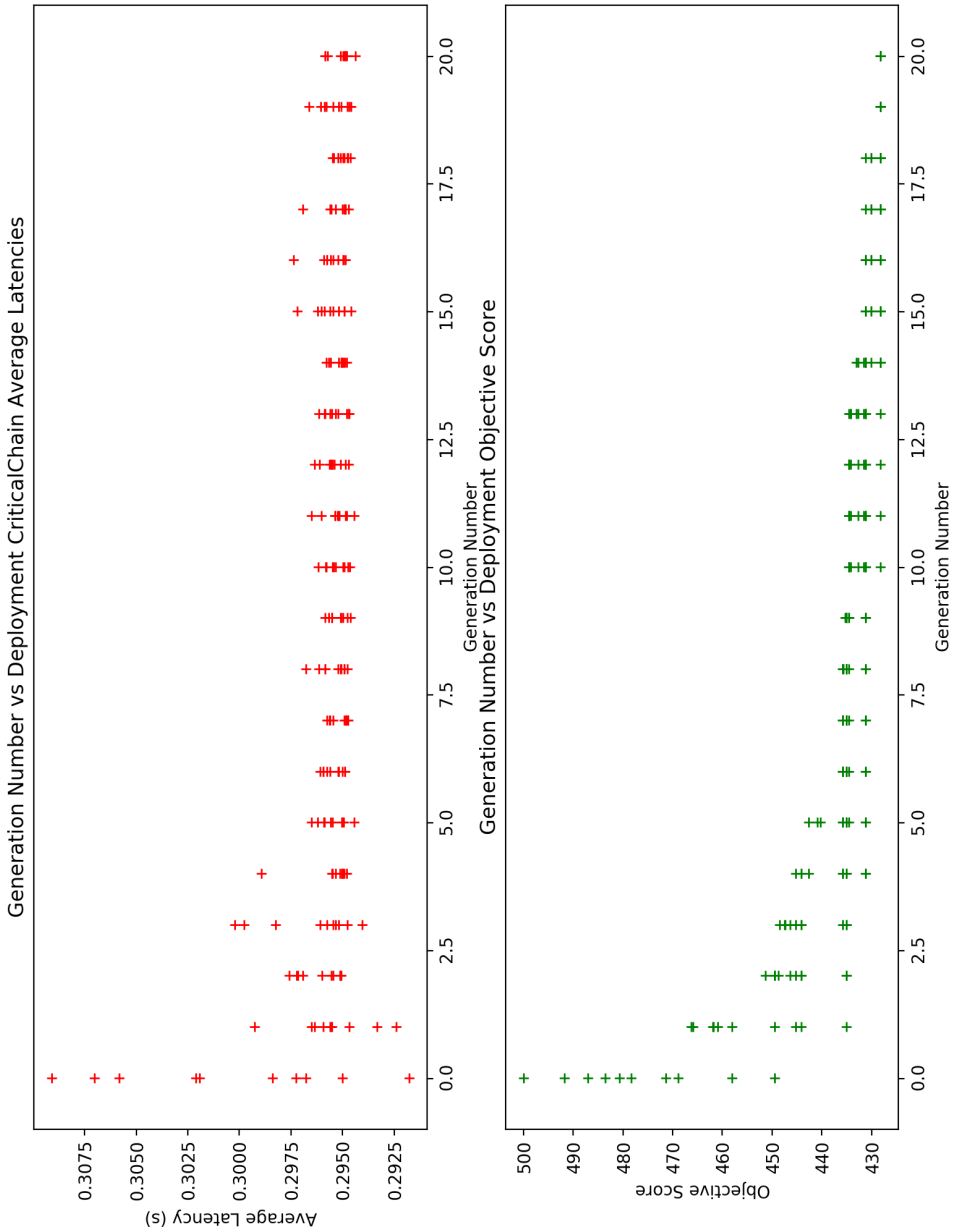

Figure D.54: 1Gb Network Configuration

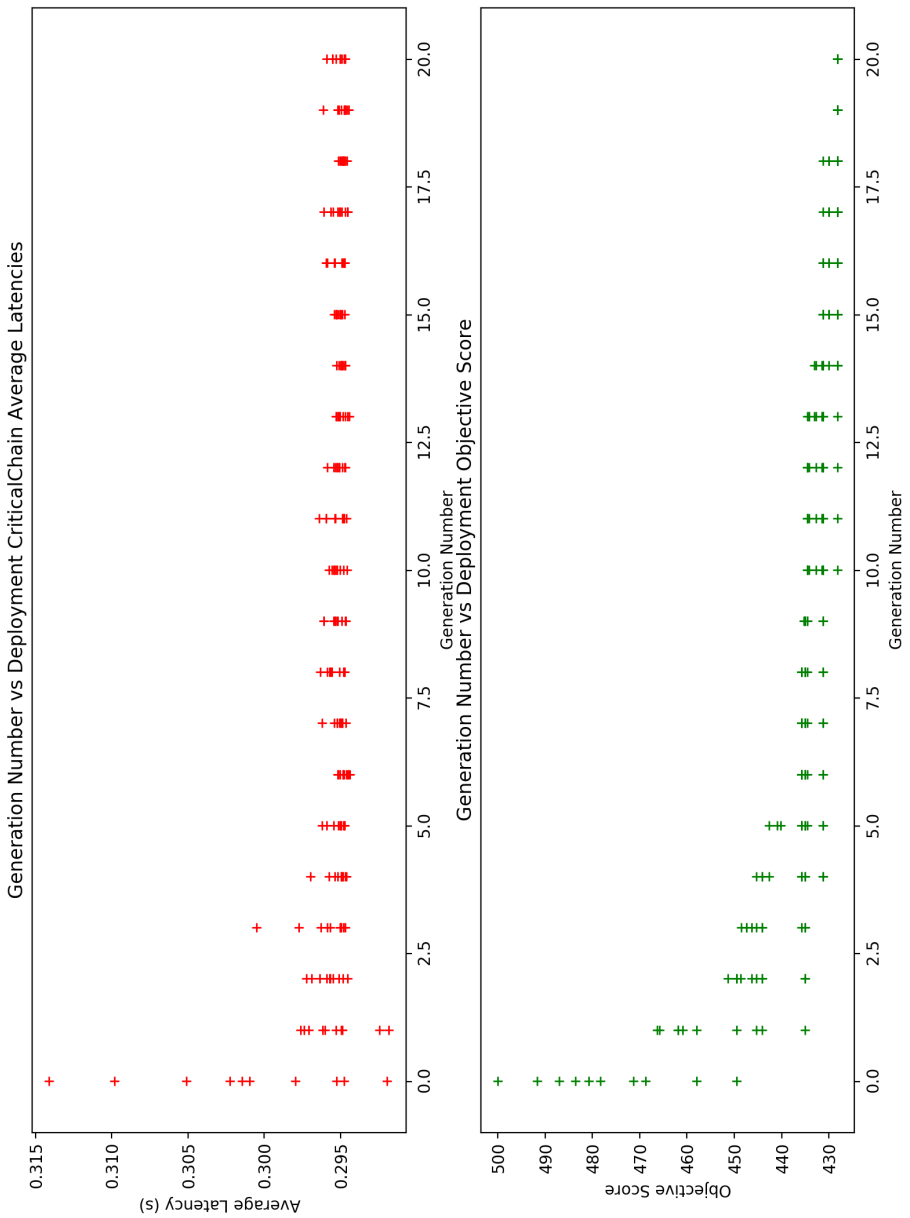

Figure D.55: 1Gb Network Configuration

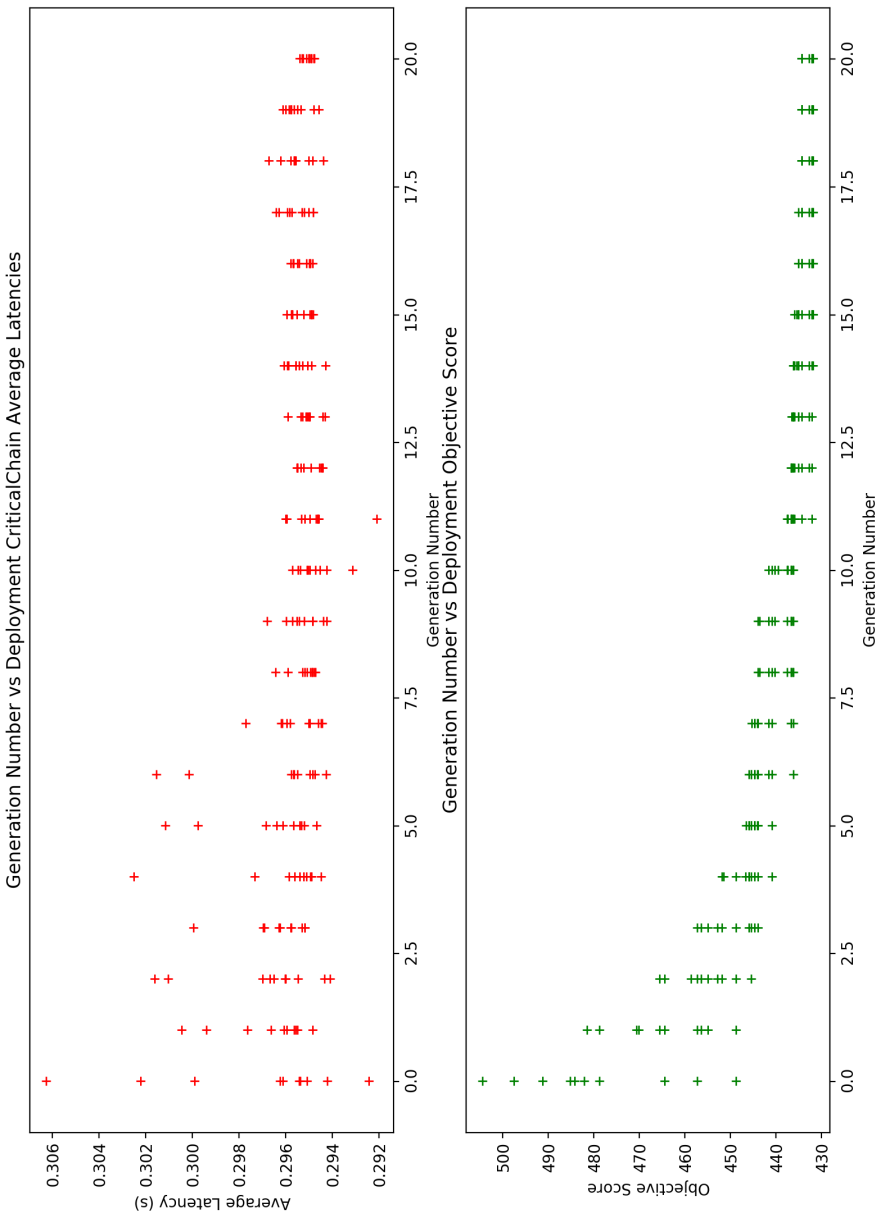

Figure D.56: 100Mb Network Configuration

D.7 Software Deployment Mandate Constraint Confirmation Text Files

D.7.1 Text File - no mandate constraints

```
0 - 1 - [3, 3, 1, 0, 2, 1, 0, 2, 2, 1] - 4252.52780735
0 - 2 - [1, 2, 1, 3, 0, 2, 3, 2, 0, 0] - 893.191068852
0 - 3 - [1, 0, 1, 3, 3, 2, 0, 0, 2, 1] - 1053.94072736
0 - 4 - [2, 1, 1, 0, 1, 2, 0, 1, 0, 1] - 1246.6385258
0 - 5 - [0, 0, 2, 2, 0, 1, 3, 0, 2, 1] - 6320.23799242
0 - 6 - [3, 2, 0, 0, 3, 3, 1, 0, 0, 2] - 863.043551712
0 - 7 - [3, 0, 1, 1, 0, 0, 2, 1, 3, 3] - 982.883866257
0 - 8 - [3, 2, 2, 3, 0, 2, 0, 0, 0, 1] - 966.874254631
0 - 9 - [2, 1, 3, 3, 0, 3, 1, 2, 2, 3] - 971.427755072
0 - 10 - [0, 2, 1, 2, 2, 2, 1, 3, 3, 2] - 979.945962673
1 - 16 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
1 - 17 - [2, 1, 3, 3, 0, 3, 1, 2, 3, 3] - 861.864510927
1 - 6 - [3, 2, 0, 0, 3, 3, 1, 0, 0, 2] - 863.043551712
1 - 2 - [1, 2, 1, 3, 0, 2, 3, 2, 0, 0] - 893.191068852
1 - 19 - [1, 0, 0, 3, 3, 1, 3, 0, 2, 1] - 928.800633736
1 - 8 - [3, 2, 2, 3, 0, 2, 0, 0, 0, 1] - 966.874254631
1 - 9 - [2, 1, 3, 3, 0, 3, 1, 2, 2, 3] - 971.427755072
1 - 10 - [0, 2, 1, 2, 2, 2, 1, 3, 3, 2] - 979.945962673
1 - 7 - [3, 0, 1, 1, 0, 0, 2, 1, 3, 3] - 982.883866257
1 - 14 - [3, 2, 0, 3, 0, 0, 2, 1, 3, 3] - 986.53480366
2 - 16 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
2 - 29 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
2 - 22 - [3, 2, 0, 0, 3, 3, 1, 0, 0, 3] - 839.957488429
2 - 17 - [2, 1, 3, 3, 0, 3, 1, 2, 3, 3] - 861.864510927
2 - 6 - [3, 2, 0, 0, 3, 3, 1, 0, 0, 2] - 863.043551712
2 - 24 - [2, 1, 0, 3, 0, 2, 0, 0, 0, 1] - 868.716229992
2 - 26 - [3, 1, 0, 3, 0, 2, 0, 1, 0, 1] - 882.851313679
2 - 2 - [1, 2, 1, 3, 0, 2, 3, 2, 0, 0] - 893.191068852
2 - 19 - [1, 0, 0, 3, 3, 1, 3, 0, 2, 1] - 928.800633736
2 - 23 - [3, 2, 3, 3, 0, 3, 1, 2, 0, 3] - 954.075066859
3 - 16 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
```

```
3 - 29 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
3 - 31 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
3 - 40 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
3 - 22 - [3, 2, 0, 0, 3, 3, 1, 0, 0, 3] - 839.957488429
3 - 34 - [2, 1, 3, 3, 0, 3, 1, 0, 3, 2] - 848.966655887
3 - 17 - [2, 1, 3, 3, 0, 3, 1, 2, 3, 3] - 861.864510927
3 - 6 - [3, 2, 0, 0, 3, 3, 1, 0, 0, 2] - 863.043551712
3 - 36 - [2, 1, 0, 0, 0, 0, 2, 1, 3, 1] - 863.092200319
3 - 24 - [2, 1, 0, 3, 0, 2, 0, 0, 0, 1] - 868.716229992
4 - 16 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
4 - 42 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
4 - 47 - [3, 0, 2, 1, 1, 0, 2, 1, 3, 1] - 760.090789518
4 - 46 - [2, 1, 0, 3, 0, 3, 1, 0, 3, 3] - 778.221289427
4 - 29 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
4 - 31 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
4 - 40 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
4 - 22 - [3, 2, 0, 0, 3, 3, 1, 0, 0, 3] - 839.957488429
4 - 34 - [2, 1, 3, 3, 0, 3, 1, 0, 3, 2] - 848.966655887
4 - 49 - [3, 1, 0, 3, 0, 2, 0, 0, 0, 1] - 852.378455594
5 - 16 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
5 - 42 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
5 - 47 - [3, 0, 2, 1, 1, 0, 2, 1, 3, 1] - 760.090789518
5 - 53 - [3, 0, 2, 2, 1, 0, 2, 1, 3, 1] - 761.294043186
5 - 46 - [2, 1, 0, 3, 0, 3, 1, 0, 3, 3] - 778.221289427
5 - 29 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
5 - 31 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
5 - 40 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
5 - 54 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
5 - 22 - [3, 2, 0, 0, 3, 3, 1, 0, 0, 3] - 839.957488429
6 - 66 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 3] - 750.414741485
6 - 16 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
6 - 42 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
6 - 64 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
6 - 47 - [3, 0, 2, 1, 1, 0, 2, 1, 3, 1] - 760.090789518
6 - 53 - [3, 0, 2, 2, 1, 0, 2, 1, 3, 1] - 761.294043186
6 - 46 - [2, 1, 0, 3, 0, 3, 1, 0, 3, 3] - 778.221289427
6 - 29 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
6 - 31 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
```

```
6 - 40 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
7 - 77 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 1] - 685.042944731
7 - 66 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 3] - 750.414741485
7 - 16 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
7 - 42 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
7 - 64 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
7 - 47 - [3, 0, 2, 1, 1, 0, 2, 1, 3, 1] - 760.090789518
7 - 53 - [3, 0, 2, 2, 1, 0, 2, 1, 3, 1] - 761.294043186
7 - 46 - [2, 1, 0, 3, 0, 3, 1, 0, 3, 3] - 778.221289427
7 - 29 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
7 - 31 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 3] - 820.314309043
8 - 77 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 1] - 685.042944731
8 - 88 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 3] - 729.698493056
8 - 66 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 3] - 750.414741485
8 - 83 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 3] - 750.414741485
8 - 16 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
8 - 42 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
8 - 64 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
8 - 47 - [3, 0, 2, 1, 1, 0, 2, 1, 3, 1] - 760.090789518
8 - 53 - [3, 0, 2, 2, 1, 0, 2, 1, 3, 1] - 761.294043186
8 - 46 - [2, 1, 0, 3, 0, 3, 1, 0, 3, 3] - 778.221289427
9 - 77 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 1] - 685.042944731
9 - 96 - [3, 0, 1, 1, 1, 2, 2, 1, 3, 1] - 695.899300559
9 - 88 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 3] - 729.698493056
9 - 66 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 3] - 750.414741485
9 - 83 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 3] - 750.414741485
9 - 16 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
9 - 42 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
9 - 64 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
9 - 47 - [3, 0, 2, 1, 1, 0, 2, 1, 3, 1] - 760.090789518
9 - 53 - [3, 0, 2, 2, 1, 0, 2, 1, 3, 1] - 761.294043186
10 - 77 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 1] - 685.042944731
10 - 107 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
10 - 96 - [3, 0, 1, 1, 1, 2, 2, 1, 3, 1] - 695.899300559
10 - 101 - [3, 0, 1, 2, 1, 0, 2, 1, 2, 1] - 697.348628595
10 - 88 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 3] - 729.698493056
10 - 106 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 0] - 747.375692458
10 - 66 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 3] - 750.414741485
```

```
10 - 83 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 3] - 750.414741485
10 - 16 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
10 - 42 - [3, 0, 1, 1, 1, 0, 2, 1, 3, 1] - 757.684282183
11 - 115 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
11 - 77 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 1] - 685.042944731
11 - 107 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
11 - 111 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
11 - 96 - [3, 0, 1, 1, 1, 2, 2, 1, 3, 1] - 695.899300559
11 - 101 - [3, 0, 1, 2, 1, 0, 2, 1, 2, 1] - 697.348628595
11 - 88 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 3] - 729.698493056
11 - 106 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 0] - 747.375692458
11 - 66 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 3] - 750.414741485
11 - 83 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 3] - 750.414741485
12 - 123 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
12 - 115 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
12 - 77 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 1] - 685.042944731
12 - 107 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
12 - 111 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
12 - 130 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
12 - 96 - [3, 0, 1, 1, 1, 2, 2, 1, 3, 1] - 695.899300559
12 - 101 - [3, 0, 1, 2, 1, 0, 2, 1, 2, 1] - 697.348628595
12 - 129 - [2, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 710.036378072
12 - 88 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 3] - 729.698493056
13 - 123 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
13 - 115 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
13 - 77 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 1] - 685.042944731
13 - 107 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
13 - 111 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
13 - 130 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
13 - 132 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
13 - 96 - [3, 0, 1, 1, 1, 2, 2, 1, 3, 1] - 695.899300559
13 - 101 - [3, 0, 1, 2, 1, 0, 2, 1, 2, 1] - 697.348628595
13 - 129 - [2, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 710.036378072
14 - 123 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
14 - 141 - [3, 0, 2, 1, 1, 1, 2, 1, 2, 1] - 652.247601064
14 - 115 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
14 - 144 - [3, 0, 1, 2, 1, 3, 2, 1, 2, 1] - 680.964086895
14 - 77 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 1] - 685.042944731
```

```
14 - 107 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
14 - 111 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
14 - 130 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
14 - 132 - [3, 0, 1, 1, 1, 1, 2, 1, 3, 1] - 694.696046892
14 - 96 - [3, 0, 1, 1, 1, 2, 2, 1, 3, 1] - 695.899300559
15 - 123 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
15 - 152 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
15 - 141 - [3, 0, 2, 1, 1, 1, 2, 1, 2, 1] - 652.247601064
15 - 115 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
15 - 156 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
15 - 157 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
15 - 153 - [3, 0, 1, 1, 1, 2, 2, 3, 2, 1] - 670.908297012
15 - 144 - [3, 0, 1, 2, 1, 3, 2, 1, 2, 1] - 680.964086895
15 - 160 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 3] - 680.964086895
15 - 77 - [3, 0, 1, 1, 1, 0, 2, 1, 2, 1] - 685.042944731
16 - 123 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
16 - 152 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
16 - 141 - [3, 0, 2, 1, 1, 1, 2, 1, 2, 1] - 652.247601064
16 - 164 - [3, 0, 2, 1, 1, 1, 2, 1, 2, 1] - 652.247601064
16 - 115 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
16 - 156 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
16 - 157 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
16 - 153 - [3, 0, 1, 1, 1, 2, 2, 3, 2, 1] - 670.908297012
16 - 144 - [3, 0, 1, 2, 1, 3, 2, 1, 2, 1] - 680.964086895
16 - 160 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 3] - 680.964086895
17 - 123 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
17 - 152 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
17 - 175 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
17 - 179 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
17 - 141 - [3, 0, 2, 1, 1, 1, 2, 1, 2, 1] - 652.247601064
17 - 164 - [3, 0, 2, 1, 1, 1, 2, 1, 2, 1] - 652.247601064
17 - 115 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
17 - 156 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
17 - 157 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
17 - 153 - [3, 0, 1, 1, 1, 2, 2, 3, 2, 1] - 670.908297012
18 - 123 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
18 - 152 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
18 - 175 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
```

```
18 - 179 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
18 - 187 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
18 - 141 - [3, 0, 2, 1, 1, 1, 2, 1, 2, 1] - 652.247601064
18 - 164 - [3, 0, 2, 1, 1, 1, 2, 1, 2, 1] - 652.247601064
18 - 115 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
18 - 156 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
18 - 157 - [3, 0, 1, 1, 1, 3, 2, 1, 2, 1] - 669.634380065
19 - 123 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
19 - 152 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
19 - 175 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
19 - 179 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
19 - 187 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
19 - 194 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
19 - 195 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
19 - 141 - [3, 0, 2, 1, 1, 1, 2, 1, 2, 1] - 652.247601064
19 - 164 - [3, 0, 2, 1, 1, 1, 2, 1, 2, 1] - 652.247601064
19 - 196 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 2] - 652.247601064
20 - 123 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
20 - 152 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
20 - 175 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
20 - 179 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
20 - 187 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
20 - 194 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
20 - 195 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
20 - 201 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
20 - 206 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
20 - 208 - [3, 0, 1, 1, 1, 2, 2, 1, 2, 1] - 640.814305461
```

D.7.2 Text File - 'componentMandateDeployGroup' mandate constraints

```
0 - 1 - [0, 3, 2, 1, 1, 3, 1, 2, 0, 1] - 23484.5557481
0 - 2 - [0, 1, 3, 1, 3, 1, 2, 1, 0, 3] - 962.595302247
0 - 3 - [0, 1, 1, 3, 0, 0, 3, 0, 3, 1] - 67902.0986374
0 - 4 - [0, 1, 0, 1, 3, 1, 3, 0, 2, 1] - 1475714.79919
```

```
0 - 5 - [0, 3, 1, 3, 1, 2, 1, 0, 3, 2] - 19964.070099
0 - 6 - [0, 3, 0, 0, 3, 0, 3, 3, 0, 2] - 23164.3999298
0 - 7 - [0, 1, 3, 2, 2, 3, 3, 0, 2, 0] - 23202.0599031
0 - 8 - [0, 2, 2, 2, 1, 1, 3, 2, 3, 3] - 243548.206235
0 - 9 - [0, 1, 2, 2, 3, 0, 3, 3, 2, 0] - 74689.6390619
0 - 10 - [0, 3, 2, 2, 1, 0, 0, 1, 2, 1] - 19701750.731
1 - 13 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
1 - 19 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
1 - 11 - [0, 1, 3, 2, 2, 3, 3, 0, 2, 0] - 739.911012671
1 - 12 - [0, 1, 3, 3, 3, 1, 2, 1, 0, 3] - 872.851964894
1 - 14 - [0, 3, 0, 3, 1, 2, 1, 0, 2, 0] - 881.408966774
1 - 2 - [0, 1, 3, 1, 3, 1, 2, 1, 0, 3] - 962.595302247
1 - 18 - [0, 3, 3, 3, 3, 1, 2, 1, 2, 0] - 994.188870304
1 - 16 - [0, 1, 3, 3, 1, 2, 1, 0, 2, 0] - 1070.15238969
1 - 20 - [0, 1, 1, 3, 0, 0, 3, 0, 2, 1] - 1091.68962969
1 - 15 - [0, 3, 0, 3, 3, 1, 2, 1, 0, 3] - 1151.62377711
2 - 13 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
2 - 19 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
2 - 11 - [0, 1, 3, 2, 2, 3, 3, 0, 2, 0] - 739.911012671
2 - 23 - [0, 1, 2, 2, 3, 0, 0, 3, 3, 2] - 792.373295338
2 - 26 - [0, 1, 3, 1, 3, 1, 2, 1, 2, 3] - 851.658017718
2 - 28 - [0, 1, 2, 2, 3, 0, 0, 3, 3, 0] - 856.593886371
2 - 12 - [0, 1, 3, 3, 3, 1, 2, 1, 0, 3] - 872.851964894
2 - 14 - [0, 3, 0, 3, 1, 2, 1, 0, 2, 0] - 881.408966774
2 - 24 - [0, 1, 3, 1, 3, 1, 2, 0, 0, 3] - 953.405404092
2 - 2 - [0, 1, 3, 1, 3, 1, 2, 1, 0, 3] - 962.595302247
3 - 13 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
3 - 19 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
3 - 11 - [0, 1, 3, 2, 2, 3, 3, 0, 2, 0] - 739.911012671
3 - 33 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 0] - 785.076131461
3 - 23 - [0, 1, 2, 2, 3, 0, 0, 3, 3, 2] - 792.373295338
3 - 31 - [0, 1, 3, 2, 2, 3, 0, 2, 3, 0] - 794.623956575
3 - 32 - [0, 1, 2, 2, 3, 0, 3, 0, 2, 0] - 796.198297677
3 - 34 - [0, 1, 2, 2, 3, 0, 3, 0, 2, 0] - 796.198297677
3 - 40 - [0, 1, 2, 2, 3, 0, 3, 0, 3, 0] - 801.936689214
3 - 36 - [0, 1, 3, 2, 2, 3, 0, 2, 2, 0] - 820.771303576
4 - 13 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
4 - 19 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
```

```
4 - 49 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
4 - 11 - [0, 1, 3, 2, 2, 3, 3, 0, 2, 0] - 739.911012671
4 - 41 - [0, 1, 2, 2, 3, 0, 3, 0, 3, 2] - 743.088947192
4 - 43 - [0, 1, 2, 2, 3, 0, 3, 0, 2, 2] - 749.933371229
4 - 45 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 0] - 769.718291011
4 - 46 - [0, 1, 2, 2, 2, 3, 0, 2, 3, 0] - 777.550516627
4 - 33 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 0] - 785.076131461
4 - 23 - [0, 1, 2, 2, 3, 0, 0, 3, 3, 2] - 792.373295338
5 - 58 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 2] - 717.220502297
5 - 13 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
5 - 19 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
5 - 49 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
5 - 11 - [0, 1, 3, 2, 2, 3, 3, 0, 2, 0] - 739.911012671
5 - 41 - [0, 1, 2, 2, 3, 0, 3, 0, 3, 2] - 743.088947192
5 - 43 - [0, 1, 2, 2, 3, 0, 3, 0, 2, 2] - 749.933371229
5 - 45 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 0] - 769.718291011
5 - 46 - [0, 1, 2, 2, 2, 3, 0, 2, 3, 0] - 777.550516627
5 - 33 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 0] - 785.076131461
6 - 58 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 2] - 717.220502297
6 - 63 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 2] - 717.220502297
6 - 13 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
6 - 19 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
6 - 49 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
6 - 11 - [0, 1, 3, 2, 2, 3, 3, 0, 2, 0] - 739.911012671
6 - 41 - [0, 1, 2, 2, 3, 0, 3, 0, 3, 2] - 743.088947192
6 - 67 - [0, 1, 2, 3, 3, 0, 3, 3, 3, 2] - 746.94568723
6 - 43 - [0, 1, 2, 2, 3, 0, 3, 0, 2, 2] - 749.933371229
6 - 61 - [0, 1, 2, 2, 3, 0, 3, 0, 2, 2] - 749.933371229
7 - 75 - [0, 1, 2, 2, 3, 3, 3, 3, 3, 2] - 697.907638567
7 - 58 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 2] - 717.220502297
7 - 63 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 2] - 717.220502297
7 - 13 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
7 - 19 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
7 - 49 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
7 - 11 - [0, 1, 3, 2, 2, 3, 3, 0, 2, 0] - 739.911012671
7 - 41 - [0, 1, 2, 2, 3, 0, 3, 0, 3, 2] - 743.088947192
7 - 67 - [0, 1, 2, 3, 3, 0, 3, 3, 3, 2] - 746.94568723
7 - 43 - [0, 1, 2, 2, 3, 0, 3, 0, 2, 2] - 749.933371229
```

```
8 - 75 - [0, 1, 2, 2, 3, 3, 3, 3, 3, 2] - 697.907638567
8 - 81 - [0, 1, 3, 2, 2, 3, 3, 3, 3, 2] - 697.907645564
8 - 87 - [0, 1, 2, 3, 3, 0, 3, 3, 2, 2] - 716.884307819
8 - 58 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 2] - 717.220502297
8 - 63 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 2] - 717.220502297
8 - 84 - [0, 1, 2, 2, 2, 0, 3, 0, 3, 2] - 718.594314618
8 - 88 - [0, 1, 2, 2, 3, 0, 3, 3, 2, 2] - 724.188909536
8 - 13 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
8 - 19 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
8 - 49 - [0, 1, 2, 2, 3, 0, 3, 3, 3, 2] - 731.86182571
9 - 97 - [0, 1, 2, 2, 3, 0, 3, 2, 3, 2] - 694.443270938
9 - 75 - [0, 1, 2, 2, 3, 3, 3, 3, 3, 2] - 697.907638567
9 - 81 - [0, 1, 3, 2, 2, 3, 3, 3, 3, 2] - 697.907645564
9 - 100 - [0, 1, 3, 2, 2, 3, 3, 3, 3, 2] - 697.907645564
9 - 87 - [0, 1, 2, 3, 3, 0, 3, 3, 2, 2] - 716.884307819
9 - 58 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 2] - 717.220502297
9 - 63 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 2] - 717.220502297
9 - 84 - [0, 1, 2, 2, 2, 0, 3, 0, 3, 2] - 718.594314618
9 - 88 - [0, 1, 2, 2, 3, 0, 3, 3, 2, 2] - 724.188909536
9 - 99 - [0, 1, 3, 2, 3, 0, 2, 2, 3, 2] - 727.810144337
10 - 102 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
10 - 109 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
10 - 97 - [0, 1, 2, 2, 3, 0, 3, 2, 3, 2] - 694.443270938
10 - 75 - [0, 1, 2, 2, 3, 3, 3, 3, 3, 2] - 697.907638567
10 - 81 - [0, 1, 3, 2, 2, 3, 3, 3, 3, 2] - 697.907645564
10 - 100 - [0, 1, 3, 2, 2, 3, 3, 3, 3, 2] - 697.907645564
10 - 87 - [0, 1, 2, 3, 3, 0, 3, 3, 2, 2] - 716.884307819
10 - 101 - [0, 1, 2, 3, 3, 0, 3, 3, 2, 2] - 716.884307819
10 - 58 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 2] - 717.220502297
10 - 63 - [0, 1, 3, 2, 3, 0, 3, 2, 3, 2] - 717.220502297
11 - 102 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
11 - 109 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
11 - 97 - [0, 1, 2, 2, 3, 0, 3, 2, 3, 2] - 694.443270938
11 - 113 - [0, 1, 2, 2, 3, 0, 3, 2, 3, 2] - 694.443270938
11 - 119 - [0, 1, 3, 2, 2, 0, 3, 2, 3, 2] - 694.443270938
11 - 75 - [0, 1, 2, 2, 3, 3, 3, 3, 3, 2] - 697.907638567
11 - 120 - [0, 1, 2, 2, 3, 3, 3, 3, 3, 2] - 697.907638567
11 - 81 - [0, 1, 3, 2, 2, 3, 3, 3, 3, 2] - 697.907645564
```

```
11 - 100 - [0, 1, 3, 2, 2, 3, 3, 3, 3, 2] - 697.907645564
11 - 118 - [0, 1, 2, 2, 3, 0, 3, 2, 2, 2] - 716.020985018
12 - 128 - [0, 1, 2, 2, 3, 3, 3, 2, 2, 2] - 669.341504173
12 - 102 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
12 - 109 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
12 - 97 - [0, 1, 2, 2, 3, 0, 3, 2, 3, 2] - 694.443270938
12 - 113 - [0, 1, 2, 2, 3, 0, 3, 2, 3, 2] - 694.443270938
12 - 119 - [0, 1, 3, 2, 2, 0, 3, 2, 3, 2] - 694.443270938
12 - 75 - [0, 1, 2, 2, 3, 3, 3, 3, 3, 2] - 697.907638567
12 - 120 - [0, 1, 2, 2, 3, 3, 3, 3, 3, 2] - 697.907638567
12 - 81 - [0, 1, 3, 2, 2, 3, 3, 3, 3, 2] - 697.907645564
12 - 100 - [0, 1, 3, 2, 2, 3, 3, 3, 3, 2] - 697.907645564
13 - 136 - [0, 1, 2, 2, 2, 3, 2, 3, 3, 2] - 641.517870761
13 - 128 - [0, 1, 2, 2, 3, 3, 3, 2, 2, 2] - 669.341504173
13 - 138 - [0, 1, 3, 2, 2, 3, 3, 2, 3, 3] - 669.693891996
13 - 132 - [0, 1, 3, 2, 2, 3, 3, 3, 2, 2] - 680.227193069
13 - 102 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
13 - 109 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
13 - 97 - [0, 1, 2, 2, 3, 0, 3, 2, 3, 2] - 694.443270938
13 - 113 - [0, 1, 2, 2, 3, 0, 3, 2, 3, 2] - 694.443270938
13 - 119 - [0, 1, 3, 2, 2, 0, 3, 2, 3, 2] - 694.443270938
13 - 75 - [0, 1, 2, 2, 3, 3, 3, 3, 3, 2] - 697.907638567
14 - 136 - [0, 1, 2, 2, 2, 3, 2, 3, 3, 2] - 641.517870761
14 - 147 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
14 - 149 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
14 - 128 - [0, 1, 2, 2, 3, 3, 3, 2, 2, 2] - 669.341504173
14 - 138 - [0, 1, 3, 2, 2, 3, 3, 2, 3, 3] - 669.693891996
14 - 132 - [0, 1, 3, 2, 2, 3, 3, 3, 2, 2] - 680.227193069
14 - 102 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
14 - 109 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
14 - 97 - [0, 1, 2, 2, 3, 0, 3, 2, 3, 2] - 694.443270938
14 - 113 - [0, 1, 2, 2, 3, 0, 3, 2, 3, 2] - 694.443270938
15 - 136 - [0, 1, 2, 2, 2, 3, 2, 3, 3, 2] - 641.517870761
15 - 157 - [0, 1, 3, 2, 3, 3, 3, 2, 2, 2] - 648.938019235
15 - 147 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
15 - 149 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
15 - 128 - [0, 1, 2, 2, 3, 3, 3, 2, 2, 2] - 669.341504173
15 - 138 - [0, 1, 3, 2, 2, 3, 3, 2, 3, 3] - 669.693891996
```

```
15 - 132 - [0, 1, 3, 2, 2, 3, 3, 3, 2, 2] - 680.227193069
15 - 102 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
15 - 109 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
15 - 97 - [0, 1, 2, 2, 3, 0, 3, 2, 3, 2] - 694.443270938
16 - 136 - [0, 1, 2, 2, 2, 3, 2, 3, 3, 2] - 641.517870761
16 - 157 - [0, 1, 3, 2, 3, 3, 3, 2, 2, 2] - 648.938019235
16 - 147 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
16 - 149 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
16 - 128 - [0, 1, 2, 2, 3, 3, 3, 2, 2, 2] - 669.341504173
16 - 138 - [0, 1, 3, 2, 2, 3, 3, 2, 3, 3] - 669.693891996
16 - 132 - [0, 1, 3, 2, 2, 3, 3, 3, 2, 2] - 680.227193069
16 - 169 - [0, 1, 2, 2, 2, 3, 3, 2, 2, 2] - 693.422307096
16 - 102 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
16 - 109 - [0, 1, 3, 2, 3, 0, 3, 2, 2, 2] - 693.825784687
17 - 136 - [0, 1, 2, 2, 2, 3, 2, 3, 3, 2] - 641.517870761
17 - 157 - [0, 1, 3, 2, 3, 3, 3, 2, 2, 2] - 648.938019235
17 - 147 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
17 - 178 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
17 - 149 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
17 - 172 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
17 - 176 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
17 - 128 - [0, 1, 2, 2, 3, 3, 3, 2, 2, 2] - 669.341504173
17 - 175 - [0, 1, 2, 2, 3, 3, 3, 2, 2, 2] - 669.341504173
17 - 138 - [0, 1, 3, 2, 2, 3, 3, 2, 3, 3] - 669.693891996
18 - 136 - [0, 1, 2, 2, 2, 3, 2, 3, 3, 2] - 641.517870761
18 - 157 - [0, 1, 3, 2, 3, 3, 3, 2, 2, 2] - 648.938019235
18 - 147 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
18 - 178 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
18 - 190 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
18 - 149 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
18 - 172 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
18 - 176 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
18 - 128 - [0, 1, 2, 2, 3, 3, 3, 2, 2, 2] - 669.341504173
18 - 175 - [0, 1, 2, 2, 3, 3, 3, 2, 2, 2] - 669.341504173
19 - 136 - [0, 1, 2, 2, 2, 3, 2, 3, 3, 2] - 641.517870761
19 - 198 - [0, 1, 2, 2, 2, 3, 2, 3, 3, 2] - 641.517870761
19 - 157 - [0, 1, 3, 2, 3, 3, 3, 2, 2, 2] - 648.938019235
19 - 192 - [0, 1, 3, 2, 3, 3, 3, 2, 2, 2] - 648.938019235
```

```
19 - 147 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
19 - 178 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
19 - 190 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
19 - 149 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
19 - 172 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
19 - 176 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
20 - 136 - [0, 1, 2, 2, 2, 3, 2, 3, 3, 2] - 641.517870761
20 - 198 - [0, 1, 2, 2, 2, 3, 2, 3, 3, 2] - 641.517870761
20 - 207 - [0, 1, 3, 2, 2, 2, 2, 3, 3, 2] - 648.049786313
20 - 157 - [0, 1, 3, 2, 3, 3, 3, 2, 2, 2] - 648.938019235
20 - 192 - [0, 1, 3, 2, 3, 3, 3, 2, 2, 2] - 648.938019235
20 - 206 - [0, 1, 3, 2, 3, 3, 3, 2, 2, 2] - 648.938019235
20 - 147 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
20 - 178 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
20 - 190 - [0, 1, 3, 2, 2, 3, 2, 3, 3, 2] - 655.305530711
20 - 149 - [0, 1, 3, 2, 3, 2, 3, 2, 2, 2] - 658.680101302
```

D.7.3 Text File - 'componentMandateDeployOnly' mandate constraints

```
0 - 1 - [0, 1, 0, 1, 2, 1, 2, 1, 2, 3] - 510003695.157
0 - 2 - [0, 2, 0, 1, 1, 2, 2, 2, 1, 3] - 450092669.447
0 - 3 - [0, 0, 2, 2, 1, 0, 1, 2, 1, 3] - 155447.916331
0 - 4 - [0, 2, 1, 0, 2, 2, 0, 0, 0, 3] - 184544654.901
0 - 5 - [0, 1, 1, 1, 0, 1, 2, 0, 0, 3] - 4263008.11716
0 - 6 - [0, 1, 0, 0, 0, 2, 2, 0, 2, 3] - 34016524.0462
0 - 7 - [0, 2, 0, 1, 2, 1, 2, 2, 2, 3] - 1346526777.83
0 - 8 - [0, 2, 2, 2, 1, 0, 2, 2, 0, 3] - 412840139.723
0 - 9 - [0, 2, 0, 2, 0, 2, 0, 0, 1, 3] - 1136978436.69
0 - 10 - [0, 2, 1, 2, 2, 1, 1, 0, 2, 3] - 144318291.946
1 - 13 - [0, 1, 2, 1, 2, 1, 2, 1, 2, 3] - 930.231483592
1 - 12 - [0, 1, 0, 1, 2, 1, 2, 0, 2, 3] - 1000.73484188
1 - 20 - [0, 2, 1, 1, 1, 2, 2, 2, 1, 3] - 1011.97455328
1 - 16 - [0, 1, 0, 1, 1, 1, 2, 0, 2, 3] - 1163.32932652
1 - 19 - [0, 0, 0, 0, 2, 2, 0, 0, 0, 3] - 14097.5490012
```

```
1 - 3 - [0, 0, 2, 2, 1, 0, 1, 2, 1, 3] - 155447.916331
1 - 5 - [0, 1, 1, 1, 0, 1, 2, 0, 0, 3] - 4263008.11716
1 - 6 - [0, 1, 0, 0, 0, 2, 2, 0, 2, 3] - 34016524.0462
1 - 10 - [0, 2, 1, 2, 2, 1, 1, 0, 2, 3] - 144318291.946
1 - 4 - [0, 2, 1, 0, 2, 2, 0, 0, 0, 3] - 184544654.901
2 - 13 - [0, 1, 2, 1, 2, 1, 2, 1, 2, 3] - 930.231483592
2 - 28 - [0, 2, 1, 0, 2, 1, 1, 1, 1, 3] - 934.556702423
2 - 30 - [0, 2, 1, 1, 1, 1, 2, 2, 1, 3] - 936.777573263
2 - 12 - [0, 1, 0, 1, 2, 1, 2, 0, 2, 3] - 1000.73484188
2 - 20 - [0, 2, 1, 1, 1, 2, 2, 2, 1, 3] - 1011.97455328
2 - 16 - [0, 1, 0, 1, 1, 1, 2, 0, 2, 3] - 1163.32932652
2 - 21 - [0, 2, 1, 0, 2, 1, 1, 0, 2, 3] - 2075.29424626
2 - 27 - [0, 2, 1, 1, 1, 2, 2, 0, 2, 3] - 2210.85363908
2 - 25 - [0, 1, 2, 1, 1, 1, 2, 2, 1, 3] - 2434.38980883
2 - 23 - [0, 0, 2, 2, 1, 0, 1, 0, 1, 3] - 5717.4935834
3 - 31 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
3 - 13 - [0, 1, 2, 1, 2, 1, 2, 1, 2, 3] - 930.231483592
3 - 28 - [0, 2, 1, 0, 2, 1, 1, 1, 1, 3] - 934.556702423
3 - 30 - [0, 2, 1, 1, 1, 1, 2, 2, 1, 3] - 936.777573263
3 - 12 - [0, 1, 0, 1, 2, 1, 2, 0, 2, 3] - 1000.73484188
3 - 20 - [0, 2, 1, 1, 1, 2, 2, 2, 1, 3] - 1011.97455328
3 - 33 - [0, 2, 1, 1, 1, 2, 2, 2, 1, 3] - 1011.97455328
3 - 16 - [0, 1, 0, 1, 1, 1, 2, 0, 2, 3] - 1163.32932652
3 - 21 - [0, 2, 1, 0, 2, 1, 1, 0, 2, 3] - 2075.29424626
3 - 38 - [0, 1, 2, 2, 1, 1, 2, 2, 1, 3] - 2182.9415174
4 - 31 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
4 - 13 - [0, 1, 2, 1, 2, 1, 2, 1, 2, 3] - 930.231483592
4 - 28 - [0, 2, 1, 0, 2, 1, 1, 1, 1, 3] - 934.556702423
4 - 30 - [0, 2, 1, 1, 1, 1, 2, 2, 1, 3] - 936.777573263
4 - 12 - [0, 1, 0, 1, 2, 1, 2, 0, 2, 3] - 1000.73484188
4 - 41 - [0, 1, 2, 2, 1, 1, 2, 2, 0, 3] - 1001.25861432
4 - 20 - [0, 2, 1, 1, 1, 2, 2, 2, 1, 3] - 1011.97455328
4 - 33 - [0, 2, 1, 1, 1, 2, 2, 2, 1, 3] - 1011.97455328
4 - 45 - [0, 2, 1, 1, 1, 2, 2, 2, 1, 3] - 1011.97455328
4 - 46 - [0, 2, 0, 1, 1, 2, 2, 2, 1, 3] - 1121.11523826
5 - 31 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
5 - 53 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
5 - 13 - [0, 1, 2, 1, 2, 1, 2, 1, 2, 3] - 930.231483592
```

```
5 - 28 - [0, 2, 1, 0, 2, 1, 1, 1, 1, 3] - 934.556702423
5 - 30 - [0, 2, 1, 1, 1, 1, 2, 2, 1, 3] - 936.777573263
5 - 57 - [0, 2, 1, 0, 1, 1, 2, 2, 1, 3] - 985.138457971
5 - 12 - [0, 1, 0, 1, 2, 1, 2, 0, 2, 3] - 1000.73484188
5 - 41 - [0, 1, 2, 2, 1, 1, 2, 2, 0, 3] - 1001.25861432
5 - 20 - [0, 2, 1, 1, 1, 2, 2, 2, 1, 3] - 1011.97455328
5 - 33 - [0, 2, 1, 1, 1, 2, 2, 2, 1, 3] - 1011.97455328
6 - 31 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
6 - 53 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
6 - 13 - [0, 1, 2, 1, 2, 1, 2, 1, 2, 3] - 930.231483592
6 - 28 - [0, 2, 1, 0, 2, 1, 1, 1, 1, 3] - 934.556702423
6 - 30 - [0, 2, 1, 1, 1, 1, 2, 2, 1, 3] - 936.777573263
6 - 68 - [0, 2, 1, 1, 1, 1, 2, 2, 1, 3] - 936.777573263
6 - 57 - [0, 2, 1, 0, 1, 1, 2, 2, 1, 3] - 985.138457971
6 - 69 - [0, 2, 1, 0, 1, 1, 2, 2, 1, 3] - 985.138457971
6 - 12 - [0, 1, 0, 1, 2, 1, 2, 0, 2, 3] - 1000.73484188
6 - 41 - [0, 1, 2, 2, 1, 1, 2, 2, 0, 3] - 1001.25861432
7 - 73 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
7 - 31 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
7 - 53 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
7 - 13 - [0, 1, 2, 1, 2, 1, 2, 1, 2, 3] - 930.231483592
7 - 28 - [0, 2, 1, 0, 2, 1, 1, 1, 1, 3] - 934.556702423
7 - 30 - [0, 2, 1, 1, 1, 1, 2, 2, 1, 3] - 936.777573263
7 - 68 - [0, 2, 1, 1, 1, 1, 2, 2, 1, 3] - 936.777573263
7 - 57 - [0, 2, 1, 0, 1, 1, 2, 2, 1, 3] - 985.138457971
7 - 69 - [0, 2, 1, 0, 1, 1, 2, 2, 1, 3] - 985.138457971
7 - 72 - [0, 2, 1, 0, 1, 1, 2, 2, 1, 3] - 985.138457971
8 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
8 - 73 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
8 - 86 - [0, 1, 2, 1, 2, 0, 2, 1, 2, 3] - 905.374772661
8 - 31 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
8 - 53 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
8 - 13 - [0, 1, 2, 1, 2, 1, 2, 1, 2, 3] - 930.231483592
8 - 28 - [0, 2, 1, 0, 2, 1, 1, 1, 1, 3] - 934.556702423
8 - 82 - [0, 2, 1, 1, 1, 2, 0, 0, 1, 3] - 936.168066857
8 - 30 - [0, 2, 1, 1, 1, 1, 2, 2, 1, 3] - 936.777573263
8 - 68 - [0, 2, 1, 1, 1, 1, 2, 2, 1, 3] - 936.777573263
9 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
```

```
9 - 100 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
9 - 73 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
9 - 86 - [0, 1, 2, 1, 2, 0, 2, 1, 2, 3] - 905.374772661
9 - 31 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
9 - 53 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
9 - 13 - [0, 1, 2, 1, 2, 1, 2, 1, 2, 3] - 930.231483592
9 - 28 - [0, 2, 1, 0, 2, 1, 1, 1, 1, 3] - 934.556702423
9 - 94 - [0, 2, 1, 0, 2, 1, 1, 1, 1, 3] - 934.556702423
9 - 82 - [0, 2, 1, 1, 1, 2, 0, 0, 1, 3] - 936.168066857
10 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
10 - 100 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
10 - 73 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
10 - 108 - [0, 2, 1, 1, 2, 1, 1, 1, 1, 3] - 895.447826962
10 - 86 - [0, 1, 2, 1, 2, 0, 2, 1, 2, 3] - 905.374772661
10 - 31 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
10 - 53 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
10 - 106 - [0, 2, 1, 1, 1, 0, 0, 0, 1, 3] - 921.844897071
10 - 13 - [0, 1, 2, 1, 2, 1, 2, 1, 2, 3] - 930.231483592
10 - 28 - [0, 2, 1, 0, 2, 1, 1, 1, 1, 3] - 934.556702423
11 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
11 - 100 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
11 - 73 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
11 - 118 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
11 - 108 - [0, 2, 1, 1, 2, 1, 1, 1, 1, 3] - 895.447826962
11 - 86 - [0, 1, 2, 1, 2, 0, 2, 1, 2, 3] - 905.374772661
11 - 31 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
11 - 53 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
11 - 106 - [0, 2, 1, 1, 1, 0, 0, 0, 1, 3] - 921.844897071
11 - 13 - [0, 1, 2, 1, 2, 1, 2, 1, 2, 3] - 930.231483592
12 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
12 - 100 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
12 - 126 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
12 - 130 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
12 - 124 - [0, 2, 1, 1, 1, 2, 1, 2, 1, 3] - 858.83170805
12 - 73 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
12 - 118 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
12 - 108 - [0, 2, 1, 1, 2, 1, 1, 1, 1, 3] - 895.447826962
12 - 86 - [0, 1, 2, 1, 2, 0, 2, 1, 2, 3] - 905.374772661
```

```
12 - 31 - [0, 2, 1, 1, 1, 2, 2, 1, 1, 3] - 917.029459676
13 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
13 - 100 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
13 - 126 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
13 - 130 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
13 - 136 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
13 - 124 - [0, 2, 1, 1, 1, 2, 1, 2, 1, 3] - 858.83170805
13 - 73 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
13 - 118 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
13 - 108 - [0, 2, 1, 1, 2, 1, 1, 1, 1, 3] - 895.447826962
13 - 86 - [0, 1, 2, 1, 2, 0, 2, 1, 2, 3] - 905.374772661
14 - 144 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
14 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
14 - 100 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
14 - 126 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
14 - 130 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
14 - 136 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
14 - 124 - [0, 2, 1, 1, 1, 2, 1, 2, 1, 3] - 858.83170805
14 - 73 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
14 - 118 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
14 - 142 - [0, 2, 1, 1, 1, 2, 0, 1, 1, 3] - 868.514180481
15 - 144 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
15 - 156 - [0, 2, 1, 0, 1, 1, 1, 2, 1, 3] - 837.436246975
15 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
15 - 100 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
15 - 126 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
15 - 130 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
15 - 136 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
15 - 152 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
15 - 153 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
15 - 124 - [0, 2, 1, 1, 1, 2, 1, 2, 1, 3] - 858.83170805
16 - 144 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
16 - 162 - [0, 2, 1, 1, 1, 1, 0, 1, 1, 3] - 807.024810838
16 - 156 - [0, 2, 1, 0, 1, 1, 1, 2, 1, 3] - 837.436246975
16 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
16 - 100 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
16 - 126 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
16 - 130 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
```

```
16 - 136 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
16 - 152 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
16 - 153 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
17 - 144 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
17 - 175 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
17 - 162 - [0, 2, 1, 1, 1, 1, 0, 1, 1, 3] - 807.024810838
17 - 156 - [0, 2, 1, 0, 1, 1, 1, 2, 1, 3] - 837.436246975
17 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
17 - 100 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
17 - 126 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
17 - 130 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
17 - 136 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
17 - 152 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
18 - 144 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
18 - 175 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
18 - 189 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
18 - 162 - [0, 2, 1, 1, 1, 1, 0, 1, 1, 3] - 807.024810838
18 - 184 - [0, 2, 1, 1, 1, 1, 0, 1, 1, 3] - 807.024810838
18 - 156 - [0, 2, 1, 0, 1, 1, 1, 2, 1, 3] - 837.436246975
18 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
18 - 100 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
18 - 126 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
18 - 130 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
19 - 144 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
19 - 175 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
19 - 189 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
19 - 162 - [0, 2, 1, 1, 1, 1, 0, 1, 1, 3] - 807.024810838
19 - 184 - [0, 2, 1, 1, 1, 1, 0, 1, 1, 3] - 807.024810838
19 - 156 - [0, 2, 1, 0, 1, 1, 1, 2, 1, 3] - 837.436246975
19 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
19 - 100 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
19 - 126 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
19 - 130 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
20 - 144 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
20 - 175 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
20 - 189 - [0, 2, 1, 1, 1, 1, 1, 1, 1, 3] - 804.190625359
20 - 207 - [0, 2, 1, 1, 1, 1, 1, 2, 1, 3] - 805.313007799
20 - 162 - [0, 2, 1, 1, 1, 1, 0, 1, 1, 3] - 807.024810838
```

```
20 - 184 - [0, 2, 1, 1, 1, 1, 0, 1, 1, 3] - 807.024810838
20 - 156 - [0, 2, 1, 0, 1, 1, 1, 2, 1, 3] - 837.436246975
20 - 201 - [0, 2, 1, 0, 1, 1, 1, 2, 1, 3] - 837.436246975
20 - 84 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
20 - 100 - [0, 2, 1, 1, 1, 2, 1, 1, 1, 3] - 846.187154959
```

D.7.4 Text File - 'noColocationPairId' mandate constraints

```
0 - 1 - [0, 2, 2, 2, 2, 0, 0, 3, 2, 1] - 2977.48951951
0 - 2 - [3, 2, 1, 0, 2, 0, 1, 3, 2, 0] - 88229.059026
0 - 3 - [3, 2, 3, 1, 0, 1, 2, 3, 1, 0] - 922.563629937
0 - 4 - [0, 0, 2, 1, 1, 2, 0, 1, 1, 3] - 5713.12260417
0 - 5 - [2, 3, 1, 3, 2, 3, 3, 0, 1, 0] - 987.754001381
0 - 6 - [2, 0, 0, 1, 0, 2, 1, 1, 0, 3] - 2437.88926956
0 - 7 - [3, 2, 1, 3, 1, 3, 0, 3, 1, 0] - 89792.3929562
0 - 8 - [1, 3, 0, 0, 1, 1, 0, 1, 1, 1] - 18157.6763419
0 - 9 - [1, 2, 3, 0, 0, 2, 2, 0, 0, 2] - 963.607564334
0 - 10 - [1, 1, 2, 3, 0, 1, 3, 0, 0, 2] - 5138.86735598
1 - 18 - [3, 2, 3, 0, 0, 0, 3, 3, 1, 0] - 873.900832551
1 - 12 - [3, 2, 3, 1, 0, 1, 3, 3, 1, 0] - 898.930316415
1 - 3 - [3, 2, 3, 1, 0, 1, 2, 3, 1, 0] - 922.563629937
1 - 14 - [1, 0, 2, 2, 2, 0, 0, 3, 2, 1] - 938.897322104
1 - 9 - [1, 2, 3, 0, 0, 2, 2, 0, 0, 2] - 963.607564334
1 - 17 - [3, 2, 3, 1, 0, 1, 2, 0, 0, 2] - 965.891776848
1 - 5 - [2, 3, 1, 3, 2, 3, 3, 0, 1, 0] - 987.754001381
1 - 20 - [3, 2, 3, 0, 0, 1, 3, 3, 0, 2] - 995.191750868
1 - 19 - [0, 2, 3, 1, 0, 1, 2, 0, 1, 0] - 1026.78434538
1 - 13 - [0, 2, 3, 0, 0, 0, 2, 0, 0, 2] - 1481.69211643
2 - 18 - [3, 2, 3, 0, 0, 0, 3, 3, 1, 0] - 873.900832551
2 - 12 - [3, 2, 3, 1, 0, 1, 3, 3, 1, 0] - 898.930316415
2 - 21 - [3, 2, 3, 0, 0, 1, 3, 3, 0, 1] - 907.922593988
2 - 3 - [3, 2, 3, 1, 0, 1, 2, 3, 1, 0] - 922.563629937
2 - 14 - [1, 0, 2, 2, 2, 0, 0, 3, 2, 1] - 938.897322104
2 - 9 - [1, 2, 3, 0, 0, 2, 2, 0, 0, 2] - 963.607564334
2 - 17 - [3, 2, 3, 1, 0, 1, 2, 0, 0, 2] - 965.891776848
2 - 26 - [1, 2, 3, 0, 0, 2, 2, 0, 1, 0] - 971.933954097
```

```
2 - 5 - [2, 3, 1, 3, 2, 3, 3, 0, 1, 0] - 987.754001381
2 - 20 - [3, 2, 3, 0, 0, 1, 3, 3, 0, 2] - 995.191750868
3 - 40 - [1, 2, 3, 0, 0, 1, 3, 3, 0, 2] - 779.535805327
3 - 35 - [2, 3, 1, 3, 0, 1, 2, 0, 0, 2] - 846.885859717
3 - 18 - [3, 2, 3, 0, 0, 0, 3, 3, 1, 0] - 873.900832551
3 - 12 - [3, 2, 3, 1, 0, 1, 3, 3, 1, 0] - 898.930316415
3 - 21 - [3, 2, 3, 0, 0, 1, 3, 3, 0, 1] - 907.922593988
3 - 31 - [2, 3, 1, 3, 0, 2, 2, 0, 0, 2] - 917.604999028
3 - 34 - [3, 2, 3, 0, 0, 1, 3, 0, 0, 2] - 921.510302745
3 - 3 - [3, 2, 3, 1, 0, 1, 2, 3, 1, 0] - 922.563629937
3 - 14 - [1, 0, 2, 2, 2, 0, 0, 3, 2, 1] - 938.897322104
3 - 9 - [1, 2, 3, 0, 0, 2, 2, 0, 0, 2] - 963.607564334
4 - 46 - [1, 2, 3, 0, 0, 0, 3, 3, 1, 0] - 749.290111711
4 - 40 - [1, 2, 3, 0, 0, 1, 3, 3, 0, 2] - 779.535805327
4 - 48 - [3, 1, 2, 2, 2, 1, 3, 3, 0, 2] - 834.980529305
4 - 35 - [2, 3, 1, 3, 0, 1, 2, 0, 0, 2] - 846.885859717
4 - 18 - [3, 2, 3, 0, 0, 0, 3, 3, 1, 0] - 873.900832551
4 - 12 - [3, 2, 3, 1, 0, 1, 3, 3, 1, 0] - 898.930316415
4 - 21 - [3, 2, 3, 0, 0, 1, 3, 3, 0, 1] - 907.922593988
4 - 31 - [2, 3, 1, 3, 0, 2, 2, 0, 0, 2] - 917.604999028
4 - 34 - [3, 2, 3, 0, 0, 1, 3, 0, 0, 2] - 921.510302745
4 - 50 - [3, 2, 3, 0, 0, 1, 3, 0, 0, 2] - 921.510302745
5 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
5 - 46 - [1, 2, 3, 0, 0, 0, 3, 3, 1, 0] - 749.290111711
5 - 40 - [1, 2, 3, 0, 0, 1, 3, 3, 0, 2] - 779.535805327
5 - 48 - [3, 1, 2, 2, 2, 1, 3, 3, 0, 2] - 834.980529305
5 - 35 - [2, 3, 1, 3, 0, 1, 2, 0, 0, 2] - 846.885859717
5 - 57 - [2, 3, 1, 3, 0, 1, 2, 0, 0, 2] - 846.885859717
5 - 18 - [3, 2, 3, 0, 0, 0, 3, 3, 1, 0] - 873.900832551
5 - 53 - [1, 2, 2, 0, 0, 0, 3, 3, 1, 0] - 889.410539589
5 - 12 - [3, 2, 3, 1, 0, 1, 3, 3, 1, 0] - 898.930316415
5 - 21 - [3, 2, 3, 0, 0, 1, 3, 3, 0, 1] - 907.922593988
6 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
6 - 67 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
6 - 46 - [1, 2, 3, 0, 0, 0, 3, 3, 1, 0] - 749.290111711
6 - 40 - [1, 2, 3, 0, 0, 1, 3, 3, 0, 2] - 779.535805327
6 - 63 - [3, 1, 2, 3, 0, 1, 2, 0, 0, 3] - 805.942474446
6 - 48 - [3, 1, 2, 2, 2, 1, 3, 3, 0, 2] - 834.980529305
```

```
6 - 35 - [2, 3, 1, 3, 0, 1, 2, 0, 0, 2] - 846.885859717
6 - 57 - [2, 3, 1, 3, 0, 1, 2, 0, 0, 2] - 846.885859717
6 - 62 - [2, 3, 1, 3, 0, 1, 2, 3, 1, 0] - 851.169031313
6 - 18 - [3, 2, 3, 0, 0, 0, 3, 3, 1, 0] - 873.900832551
7 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
7 - 67 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
7 - 77 - [3, 1, 2, 3, 0, 0, 2, 3, 0, 2] - 729.892973914
7 - 46 - [1, 2, 3, 0, 0, 0, 3, 3, 1, 0] - 749.290111711
7 - 40 - [1, 2, 3, 0, 0, 1, 3, 3, 0, 2] - 779.535805327
7 - 72 - [1, 3, 1, 3, 0, 0, 2, 0, 0, 2] - 801.350614628
7 - 63 - [3, 1, 2, 3, 0, 1, 2, 0, 0, 3] - 805.942474446
7 - 48 - [3, 1, 2, 2, 2, 1, 3, 3, 0, 2] - 834.980529305
7 - 78 - [3, 1, 2, 2, 2, 1, 3, 0, 2, 2] - 835.007280708
7 - 80 - [3, 1, 2, 3, 0, 1, 2, 0, 3, 2] - 841.693354948
8 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
8 - 67 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
8 - 85 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
8 - 77 - [3, 1, 2, 3, 0, 0, 2, 3, 0, 2] - 729.892973914
8 - 46 - [1, 2, 3, 0, 0, 0, 3, 3, 1, 0] - 749.290111711
8 - 81 - [3, 1, 2, 3, 0, 2, 2, 0, 3, 2] - 767.994817361
8 - 40 - [1, 2, 3, 0, 0, 1, 3, 3, 0, 2] - 779.535805327
8 - 88 - [3, 1, 2, 2, 3, 1, 2, 0, 0, 2] - 784.306089244
8 - 86 - [3, 1, 2, 3, 0, 2, 0, 0, 3, 2] - 791.050271363
8 - 72 - [1, 3, 1, 3, 0, 0, 2, 0, 0, 2] - 801.350614628
9 - 99 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
9 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
9 - 67 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
9 - 85 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
9 - 92 - [3, 1, 2, 2, 0, 2, 2, 0, 3, 2] - 712.743588304
9 - 100 - [3, 1, 2, 3, 0, 2, 2, 2, 0, 2] - 715.109430289
9 - 96 - [3, 1, 0, 3, 0, 0, 2, 0, 0, 2] - 722.921750173
9 - 91 - [3, 1, 2, 3, 0, 2, 0, 0, 2, 2] - 725.233480949
9 - 77 - [3, 1, 2, 3, 0, 0, 2, 3, 0, 2] - 729.892973914
9 - 46 - [1, 2, 3, 0, 0, 0, 3, 3, 1, 0] - 749.290111711
10 - 99 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
10 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
10 - 67 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
10 - 85 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
```

```
10 - 92 - [3, 1, 2, 2, 0, 2, 2, 0, 3, 2] - 712.743588304
10 - 100 - [3, 1, 2, 3, 0, 2, 2, 2, 0, 2] - 715.109430289
10 - 96 - [3, 1, 0, 3, 0, 0, 2, 0, 0, 2] - 722.921750173
10 - 91 - [3, 1, 2, 3, 0, 2, 0, 0, 2, 2] - 725.233480949
10 - 77 - [3, 1, 2, 3, 0, 0, 2, 3, 0, 2] - 729.892973914
10 - 105 - [3, 1, 2, 3, 0, 2, 2, 3, 0, 0] - 729.892973914
11 - 99 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
11 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
11 - 67 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
11 - 85 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
11 - 92 - [3, 1, 2, 2, 0, 2, 2, 0, 3, 2] - 712.743588304
11 - 115 - [3, 1, 2, 2, 0, 2, 2, 0, 3, 2] - 712.743588304
11 - 118 - [3, 1, 2, 2, 0, 2, 2, 0, 3, 2] - 712.743588304
11 - 100 - [3, 1, 2, 3, 0, 2, 2, 2, 0, 2] - 715.109430289
11 - 119 - [3, 1, 2, 2, 0, 2, 3, 0, 0, 2] - 720.518279316
11 - 96 - [3, 1, 0, 3, 0, 0, 2, 0, 0, 2] - 722.921750173
12 - 99 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
12 - 122 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
12 - 127 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
12 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
12 - 67 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
12 - 85 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
12 - 123 - [3, 1, 2, 3, 0, 0, 2, 0, 0, 2] - 701.315102495
12 - 125 - [3, 1, 2, 3, 2, 2, 2, 2, 0, 2] - 707.649529662
12 - 130 - [3, 1, 2, 3, 2, 2, 2, 2, 0, 2] - 707.649529662
12 - 92 - [3, 1, 2, 2, 0, 2, 2, 0, 3, 2] - 712.743588304
13 - 99 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
13 - 122 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
13 - 127 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
13 - 133 - [3, 1, 0, 2, 0, 2, 2, 0, 0, 2] - 670.683176254
13 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
13 - 67 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
13 - 85 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
13 - 132 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
13 - 123 - [3, 1, 2, 3, 0, 0, 2, 0, 0, 2] - 701.315102495
13 - 125 - [3, 1, 2, 3, 2, 2, 2, 2, 0, 2] - 707.649529662
14 - 99 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
14 - 122 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
```

```
14 - 127 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
14 - 133 - [3, 1, 0, 2, 0, 2, 2, 0, 0, 2] - 670.683176254
14 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
14 - 67 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
14 - 85 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
14 - 132 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
14 - 141 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
14 - 146 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
15 - 99 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
15 - 122 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
15 - 127 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
15 - 153 - [3, 1, 2, 2, 0, 0, 2, 0, 0, 2] - 661.111259957
15 - 133 - [3, 1, 0, 2, 0, 2, 2, 0, 0, 2] - 670.683176254
15 - 154 - [3, 1, 2, 3, 2, 2, 2, 0, 0, 2] - 675.452536694
15 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
15 - 67 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
15 - 85 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
15 - 132 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
16 - 99 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
16 - 122 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
16 - 127 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
16 - 153 - [3, 1, 2, 2, 0, 0, 2, 0, 0, 2] - 661.111259957
16 - 133 - [3, 1, 0, 2, 0, 2, 2, 0, 0, 2] - 670.683176254
16 - 154 - [3, 1, 2, 3, 2, 2, 2, 0, 0, 2] - 675.452536694
16 - 164 - [3, 1, 0, 2, 0, 0, 2, 0, 0, 2] - 681.128079978
16 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
16 - 67 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
16 - 85 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
17 - 99 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
17 - 122 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
17 - 127 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
17 - 174 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
17 - 153 - [3, 1, 2, 2, 0, 0, 2, 0, 0, 2] - 661.111259957
17 - 133 - [3, 1, 0, 2, 0, 2, 2, 0, 0, 2] - 670.683176254
17 - 154 - [3, 1, 2, 3, 2, 2, 2, 0, 0, 2] - 675.452536694
17 - 175 - [3, 1, 2, 3, 2, 2, 2, 0, 0, 2] - 675.452536694
17 - 164 - [3, 1, 0, 2, 0, 0, 2, 0, 0, 2] - 681.128079978
17 - 52 - [3, 1, 2, 3, 0, 2, 2, 0, 0, 2] - 691.821259799
```

```
18 - 99 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
18 - 122 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
18 - 127 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
18 - 174 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
18 - 153 - [3, 1, 2, 2, 0, 0, 2, 0, 0, 2] - 661.111259957
18 - 183 - [3, 1, 2, 2, 0, 0, 2, 0, 0, 2] - 661.111259957
18 - 133 - [3, 1, 0, 2, 0, 2, 2, 0, 0, 2] - 670.683176254
18 - 187 - [3, 1, 0, 2, 0, 2, 2, 0, 0, 2] - 670.683176254
18 - 154 - [3, 1, 2, 3, 2, 2, 2, 0, 0, 2] - 675.452536694
18 - 175 - [3, 1, 2, 3, 2, 2, 2, 0, 0, 2] - 675.452536694
19 - 200 - [3, 1, 2, 2, 2, 0, 2, 0, 0, 2] - 644.586389651
19 - 99 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
19 - 122 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
19 - 127 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
19 - 174 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
19 - 153 - [3, 1, 2, 2, 0, 0, 2, 0, 0, 2] - 661.111259957
19 - 183 - [3, 1, 2, 2, 0, 0, 2, 0, 0, 2] - 661.111259957
19 - 133 - [3, 1, 0, 2, 0, 2, 2, 0, 0, 2] - 670.683176254
19 - 187 - [3, 1, 0, 2, 0, 2, 2, 0, 0, 2] - 670.683176254
19 - 191 - [3, 1, 2, 0, 0, 0, 2, 0, 0, 2] - 670.683176254
20 - 200 - [3, 1, 2, 2, 2, 0, 2, 0, 0, 2] - 644.586389651
20 - 206 - [3, 1, 2, 2, 2, 0, 2, 0, 0, 2] - 644.586389651
20 - 99 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
20 - 122 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
20 - 127 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
20 - 174 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
20 - 203 - [3, 1, 2, 2, 0, 2, 2, 0, 0, 2] - 652.41233109
20 - 153 - [3, 1, 2, 2, 0, 0, 2, 0, 0, 2] - 661.111259957
20 - 183 - [3, 1, 2, 2, 0, 0, 2, 0, 0, 2] - 661.111259957
20 - 133 - [3, 1, 0, 2, 0, 2, 2, 0, 0, 2] - 670.683176254
```

D.7.5 Text File - deployment mandate multi-constraints

```
0 - 1 - [0, 0, 1, 0, 1, 2, 0, 2, 1, 3] - 56825798.9281
0 - 2 - [0, 2, 0, 1, 0, 1, 0, 2, 2, 3] - 74248857.1131
0 - 3 - [0, 1, 0, 1, 0, 1, 1, 0, 0, 3] - 1655261897.62
```

```
0 - 4 - [0, 0, 2, 0, 1, 0, 2, 0, 1, 3] - 24382832.7063
0 - 5 - [0, 1, 2, 2, 2, 1, 1, 0, 2, 3] - 358724387.267
0 - 6 - [0, 2, 2, 0, 0, 0, 0, 2, 1, 3] - 305914950.345
0 - 7 - [0, 0, 1, 0, 2, 0, 2, 0, 1, 3] - 1556544054.58
0 - 8 - [0, 2, 0, 2, 0, 0, 0, 1, 2, 3] - 143006269.558
0 - 9 - [0, 1, 0, 1, 2, 0, 1, 2, 0, 3] - 88285283.6008
0 - 10 - [0, 0, 0, 0, 1, 0, 2, 0, 1, 3] - 3465876177.99
1 - 16 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
1 - 18 - [0, 2, 2, 0, 0, 0, 0, 2, 1, 3] - 1268.06327426
1 - 15 - [0, 0, 1, 0, 2, 1, 1, 0, 2, 3] - 5618.26894253
1 - 12 - [0, 0, 2, 0, 1, 0, 2, 1, 1, 3] - 5666.15810672
1 - 13 - [0, 0, 1, 0, 2, 0, 2, 1, 0, 3] - 9152.15154627
1 - 4 - [0, 0, 2, 0, 1, 0, 2, 0, 1, 3] - 24382832.7063
1 - 1 - [0, 0, 1, 0, 1, 2, 0, 2, 1, 3] - 56825798.9281
1 - 2 - [0, 2, 0, 1, 0, 1, 0, 2, 2, 3] - 74248857.1131
1 - 9 - [0, 1, 0, 1, 2, 0, 1, 2, 0, 3] - 88285283.6008
1 - 8 - [0, 2, 0, 2, 0, 0, 0, 1, 2, 3] - 143006269.558
2 - 16 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
2 - 18 - [0, 2, 2, 0, 0, 0, 0, 2, 1, 3] - 1268.06327426
2 - 24 - [0, 1, 1, 0, 2, 1, 0, 2, 1, 3] - 2482.62180121
2 - 27 - [0, 0, 2, 0, 1, 1, 2, 1, 1, 3] - 5073.209387
2 - 15 - [0, 0, 1, 0, 2, 1, 1, 0, 2, 3] - 5618.26894253
2 - 12 - [0, 0, 2, 0, 1, 0, 2, 1, 1, 3] - 5666.15810672
2 - 22 - [0, 0, 1, 0, 2, 1, 2, 1, 0, 3] - 8200.51817101
2 - 13 - [0, 0, 1, 0, 2, 0, 2, 1, 0, 3] - 9152.15154627
2 - 4 - [0, 0, 2, 0, 1, 0, 2, 0, 1, 3] - 24382832.7063
2 - 1 - [0, 0, 1, 0, 1, 2, 0, 2, 1, 3] - 56825798.9281
3 - 31 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
3 - 16 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
3 - 18 - [0, 2, 2, 0, 0, 0, 0, 2, 1, 3] - 1268.06327426
3 - 24 - [0, 1, 1, 0, 2, 1, 0, 2, 1, 3] - 2482.62180121
3 - 40 - [0, 2, 2, 0, 2, 1, 1, 0, 2, 3] - 2577.39585632
3 - 27 - [0, 0, 2, 0, 1, 1, 2, 1, 1, 3] - 5073.209387
3 - 15 - [0, 0, 1, 0, 2, 1, 1, 0, 2, 3] - 5618.26894253
3 - 12 - [0, 0, 2, 0, 1, 0, 2, 1, 1, 3] - 5666.15810672
3 - 36 - [0, 0, 1, 0, 2, 1, 0, 2, 1, 3] - 6345.10617152
3 - 38 - [0, 0, 1, 0, 1, 0, 0, 1, 1, 3] - 7184.8074403
4 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
```

```
4 - 31 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
4 - 16 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
4 - 46 - [0, 1, 1, 2, 2, 2, 2, 1, 0, 3] - 976.681411615
4 - 41 - [0, 1, 1, 2, 2, 1, 2, 1, 0, 3] - 1072.07290711
4 - 18 - [0, 2, 2, 0, 0, 0, 0, 2, 1, 3] - 1268.06327426
4 - 50 - [0, 2, 2, 0, 0, 0, 0, 2, 1, 3] - 1268.06327426
4 - 47 - [0, 1, 2, 0, 2, 1, 0, 0, 1, 3] - 2113.53963164
4 - 24 - [0, 1, 1, 0, 2, 1, 0, 2, 1, 3] - 2482.62180121
4 - 40 - [0, 2, 2, 0, 2, 1, 1, 0, 2, 3] - 2577.39585632
5 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
5 - 31 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
5 - 52 - [0, 1, 2, 2, 2, 0, 2, 1, 0, 3] - 882.584531592
5 - 16 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
5 - 46 - [0, 1, 1, 2, 2, 2, 2, 1, 0, 3] - 976.681411615
5 - 41 - [0, 1, 1, 2, 2, 1, 2, 1, 0, 3] - 1072.07290711
5 - 51 - [0, 1, 1, 2, 1, 2, 2, 2, 0, 3] - 1094.95757855
5 - 18 - [0, 2, 2, 0, 0, 0, 0, 2, 1, 3] - 1268.06327426
5 - 50 - [0, 2, 2, 0, 0, 0, 0, 2, 1, 3] - 1268.06327426
5 - 57 - [0, 1, 2, 2, 2, 0, 0, 2, 1, 3] - 1755.31662465
6 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
6 - 31 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
6 - 67 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
6 - 52 - [0, 1, 2, 2, 2, 0, 2, 1, 0, 3] - 882.584531592
6 - 16 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
6 - 46 - [0, 1, 1, 2, 2, 2, 2, 1, 0, 3] - 976.681411615
6 - 41 - [0, 1, 1, 2, 2, 1, 2, 1, 0, 3] - 1072.07290711
6 - 51 - [0, 1, 1, 2, 1, 2, 2, 2, 0, 3] - 1094.95757855
6 - 18 - [0, 2, 2, 0, 0, 0, 0, 2, 1, 3] - 1268.06327426
6 - 50 - [0, 2, 2, 0, 0, 0, 0, 2, 1, 3] - 1268.06327426
7 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
7 - 31 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
7 - 67 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
7 - 52 - [0, 1, 2, 2, 2, 0, 2, 1, 0, 3] - 882.584531592
7 - 16 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
7 - 46 - [0, 1, 1, 2, 2, 2, 2, 1, 0, 3] - 976.681411615
7 - 75 - [0, 1, 0, 2, 2, 1, 2, 1, 0, 3] - 1056.14214258
7 - 73 - [0, 1, 1, 0, 2, 2, 2, 1, 0, 3] - 1064.09677178
7 - 76 - [0, 1, 1, 0, 2, 2, 2, 1, 0, 3] - 1064.09677178
```

```
7 - 41 - [0, 1, 1, 2, 2, 1, 2, 1, 0, 3] - 1072.07290711
8 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
8 - 31 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
8 - 67 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
8 - 52 - [0, 1, 2, 2, 2, 0, 2, 1, 0, 3] - 882.584531592
8 - 16 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
8 - 46 - [0, 1, 1, 2, 2, 2, 2, 1, 0, 3] - 976.681411615
8 - 84 - [0, 1, 0, 2, 2, 1, 2, 2, 0, 3] - 991.315465758
8 - 87 - [0, 1, 0, 2, 2, 1, 2, 2, 0, 3] - 991.315465758
8 - 75 - [0, 1, 0, 2, 2, 1, 2, 1, 0, 3] - 1056.14214258
8 - 89 - [0, 1, 0, 2, 2, 1, 2, 1, 0, 3] - 1056.14214258
9 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
9 - 31 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
9 - 67 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
9 - 91 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
9 - 52 - [0, 1, 2, 2, 2, 0, 2, 1, 0, 3] - 882.584531592
9 - 16 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
9 - 94 - [0, 1, 0, 2, 2, 2, 2, 0, 0, 3] - 958.505720195
9 - 46 - [0, 1, 1, 2, 2, 2, 2, 1, 0, 3] - 976.681411615
9 - 95 - [0, 1, 1, 2, 2, 2, 2, 1, 0, 3] - 976.681411615
9 - 84 - [0, 1, 0, 2, 2, 1, 2, 2, 0, 3] - 991.315465758
10 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
10 - 107 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
10 - 31 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
10 - 67 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
10 - 91 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
10 - 52 - [0, 1, 2, 2, 2, 0, 2, 1, 0, 3] - 882.584531592
10 - 16 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
10 - 104 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
10 - 94 - [0, 1, 0, 2, 2, 2, 2, 0, 0, 3] - 958.505720195
10 - 101 - [0, 1, 2, 1, 2, 0, 2, 1, 0, 3] - 969.130805344
11 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
11 - 107 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
11 - 31 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
11 - 67 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
11 - 91 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
11 - 114 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
11 - 52 - [0, 1, 2, 2, 2, 0, 2, 1, 0, 3] - 882.584531592
```

```
11 - 16 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
11 - 104 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
11 - 118 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
12 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
12 - 107 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
12 - 31 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
12 - 67 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
12 - 91 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
12 - 114 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
12 - 121 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
12 - 130 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
12 - 52 - [0, 1, 2, 2, 2, 0, 2, 1, 0, 3] - 882.584531592
12 - 16 - [0, 1, 2, 2, 2, 1, 2, 1, 0, 3] - 889.709608808
13 - 135 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
13 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
13 - 107 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
13 - 134 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
13 - 137 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
13 - 31 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
13 - 67 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
13 - 91 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
13 - 114 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
13 - 121 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
14 - 135 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
14 - 141 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
14 - 143 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
14 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
14 - 107 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
14 - 134 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
14 - 137 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
14 - 144 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
14 - 145 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
14 - 31 - [0, 1, 2, 2, 2, 1, 2, 2, 0, 3] - 845.119763919
15 - 135 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
15 - 141 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
15 - 143 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
15 - 157 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
15 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
```

```
15 - 107 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
15 - 134 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
15 - 137 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
15 - 144 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
15 - 145 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
16 - 135 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
16 - 141 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
16 - 143 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
16 - 157 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
16 - 163 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
16 - 162 - [0, 1, 2, 2, 2, 2, 2, 0, 0, 3] - 805.955216592
16 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
16 - 169 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
16 - 107 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
16 - 134 - [0, 1, 2, 2, 2, 0, 2, 2, 0, 3] - 839.988494686
17 - 135 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
17 - 141 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
17 - 143 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
17 - 157 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
17 - 163 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
17 - 178 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
17 - 162 - [0, 1, 2, 2, 2, 2, 2, 0, 0, 3] - 805.955216592
17 - 179 - [0, 1, 2, 2, 2, 2, 2, 0, 0, 3] - 805.955216592
17 - 45 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
17 - 169 - [0, 1, 2, 2, 2, 2, 2, 1, 0, 3] - 811.157498684
18 - 135 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
18 - 141 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
18 - 143 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
18 - 157 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
18 - 163 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
18 - 178 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
18 - 182 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
18 - 184 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
18 - 162 - [0, 1, 2, 2, 2, 2, 2, 0, 0, 3] - 805.955216592
18 - 179 - [0, 1, 2, 2, 2, 2, 2, 0, 0, 3] - 805.955216592
19 - 135 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
19 - 141 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
19 - 143 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
```

```
19 - 157 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
19 - 163 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
19 - 178 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
19 - 182 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
19 - 184 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
19 - 191 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
19 - 197 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
20 - 135 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
20 - 141 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
20 - 143 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
20 - 157 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
20 - 163 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
20 - 178 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
20 - 182 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
20 - 184 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
20 - 191 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
20 - 197 - [0, 1, 2, 2, 2, 2, 2, 2, 0, 3] - 778.679877744
```

E. Case Study MEDEA Model Diagrams

E.1 Undersea Sensor System Software Component Definitions

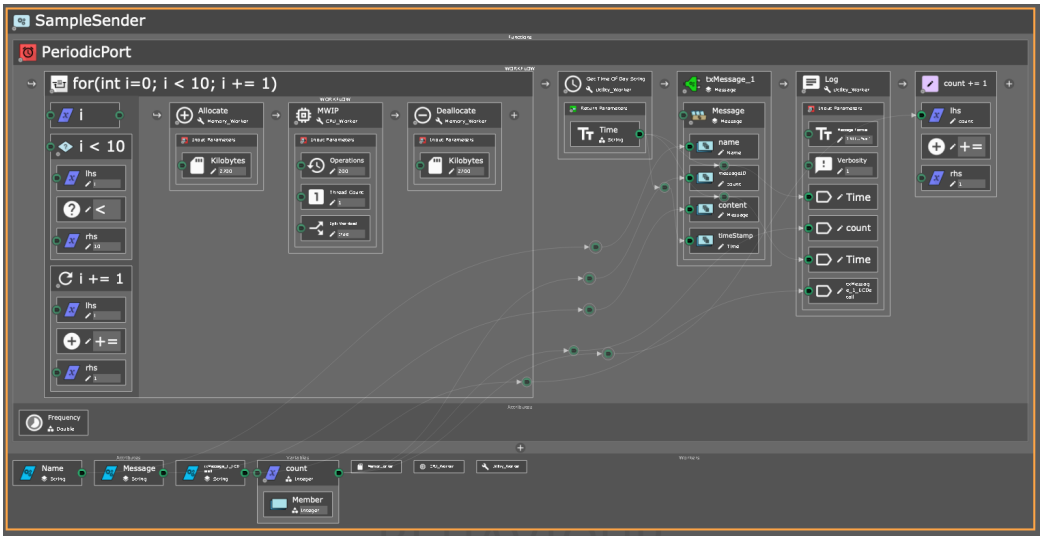

Figure E.1: Sample sender behaviour definition

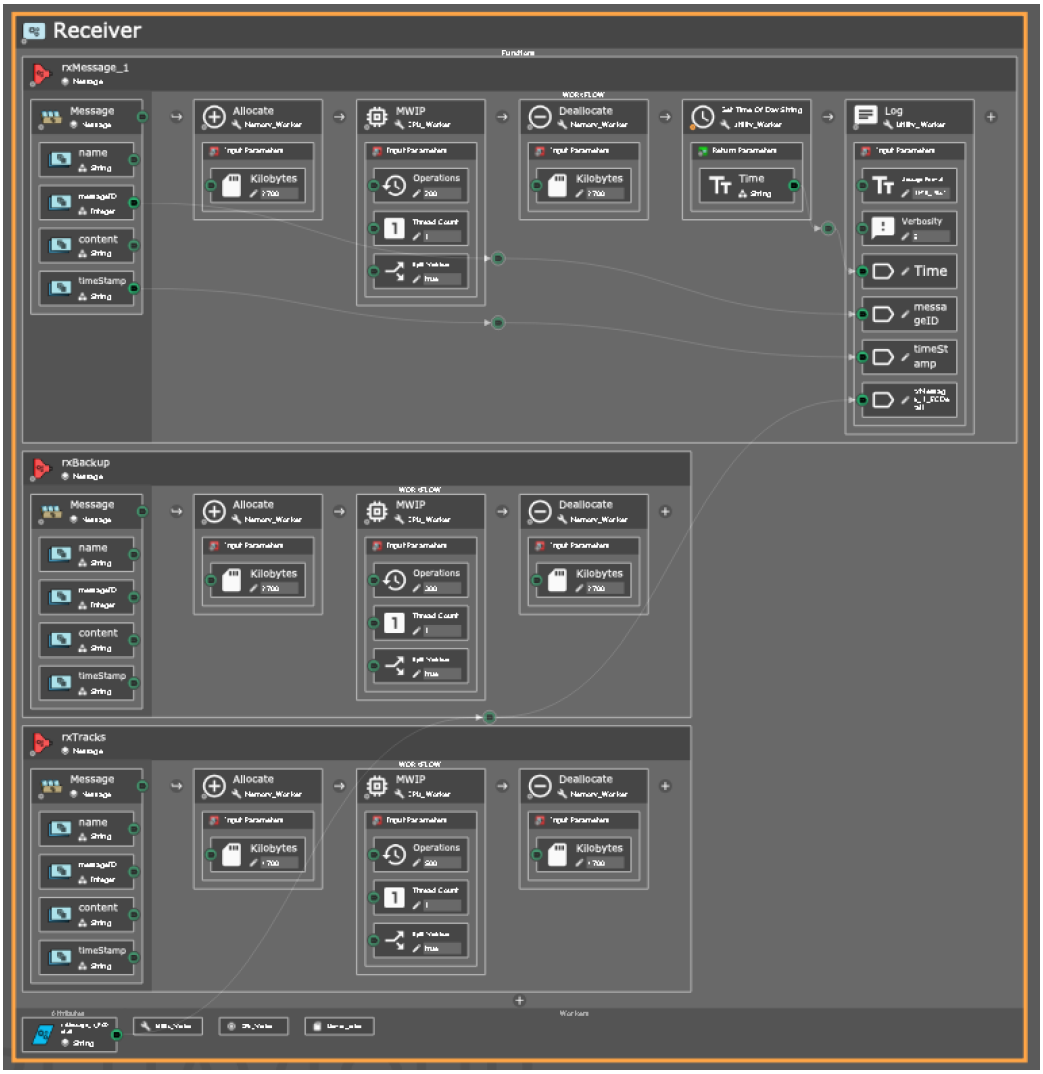

Figure E.2: Receiver behaviour definition

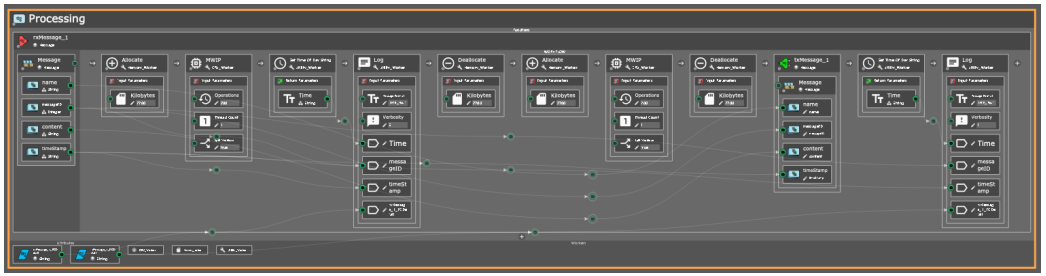

Figure E.3: Processing behaviour definition

Figure E.4: Tx_Repeater behaviour definition

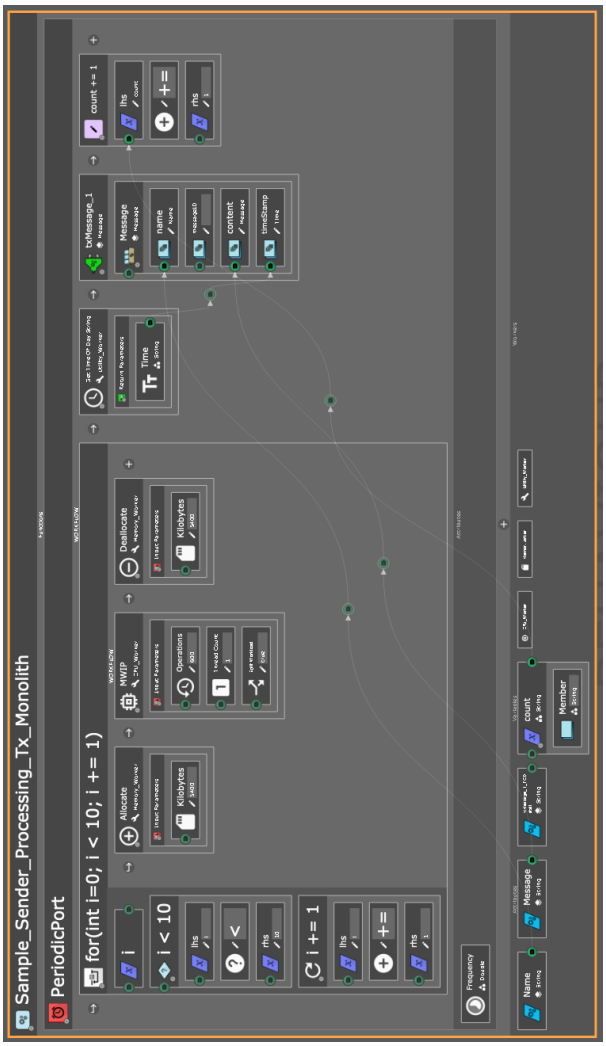

Figure E.5: Sample_Sender_processing_Tx_Monolith behaviour definition

E.2 Case Study 0: CPU and Memory Consumption Plots

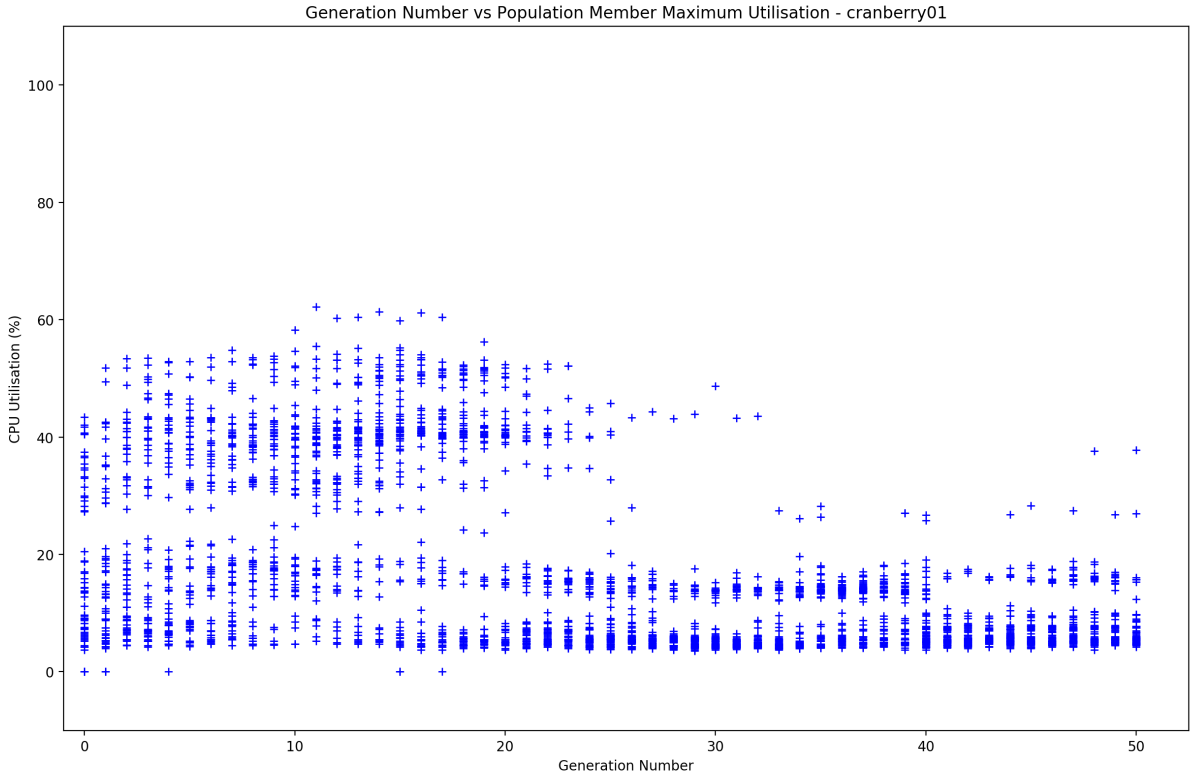

Figure E.6: Cranberry01 maximum CPU resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

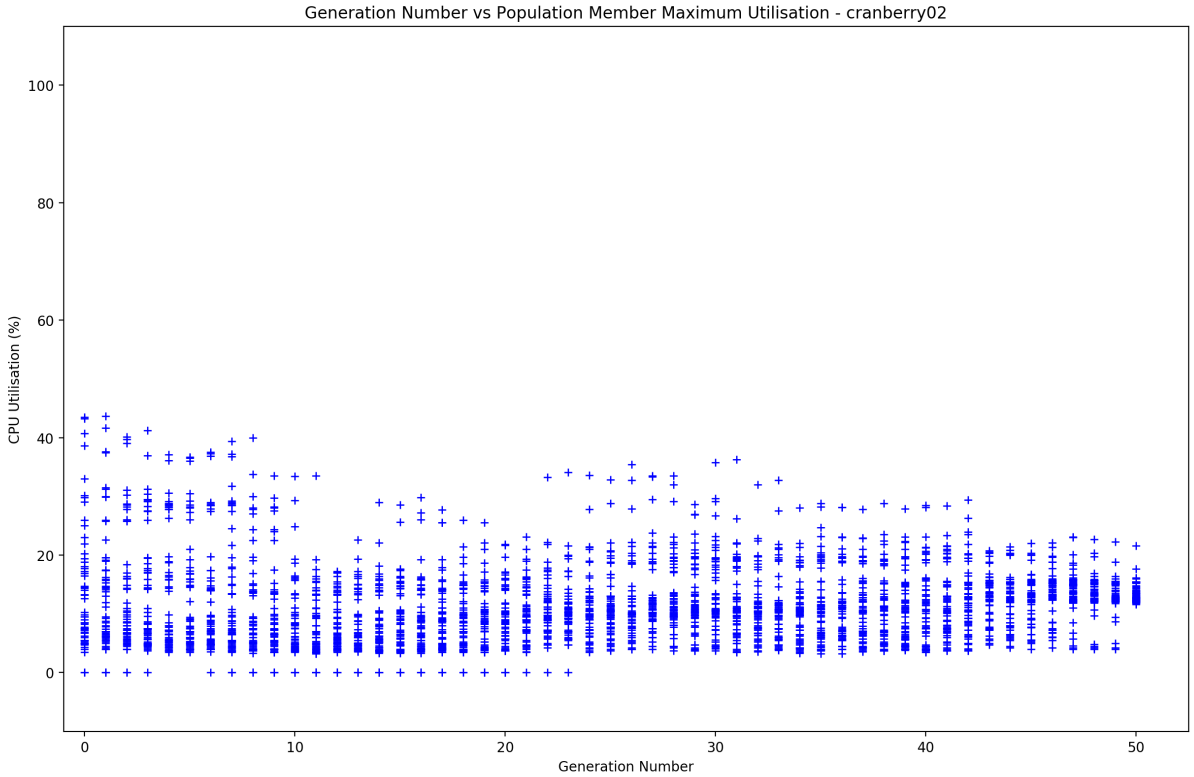

Figure E.7: Cranberry02 maximum CPU resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

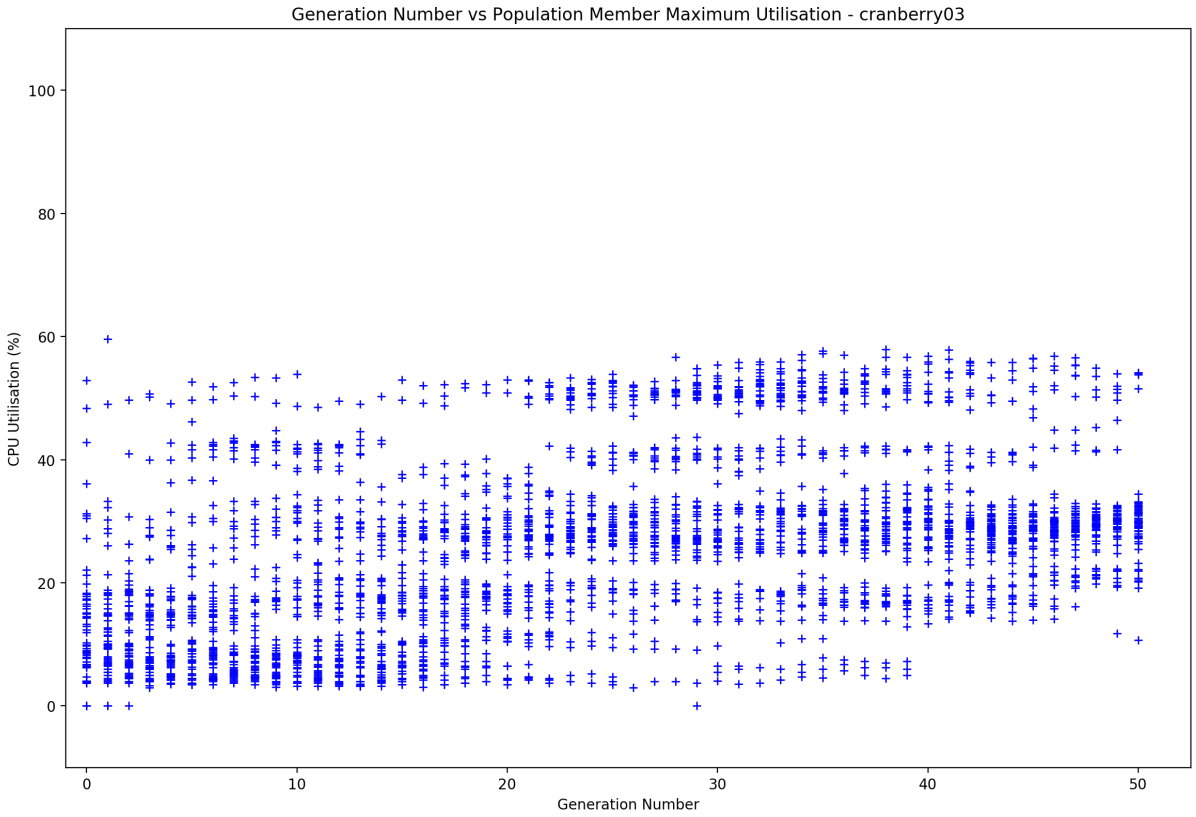

Figure E.8: Cranberry03 maximum CPU resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

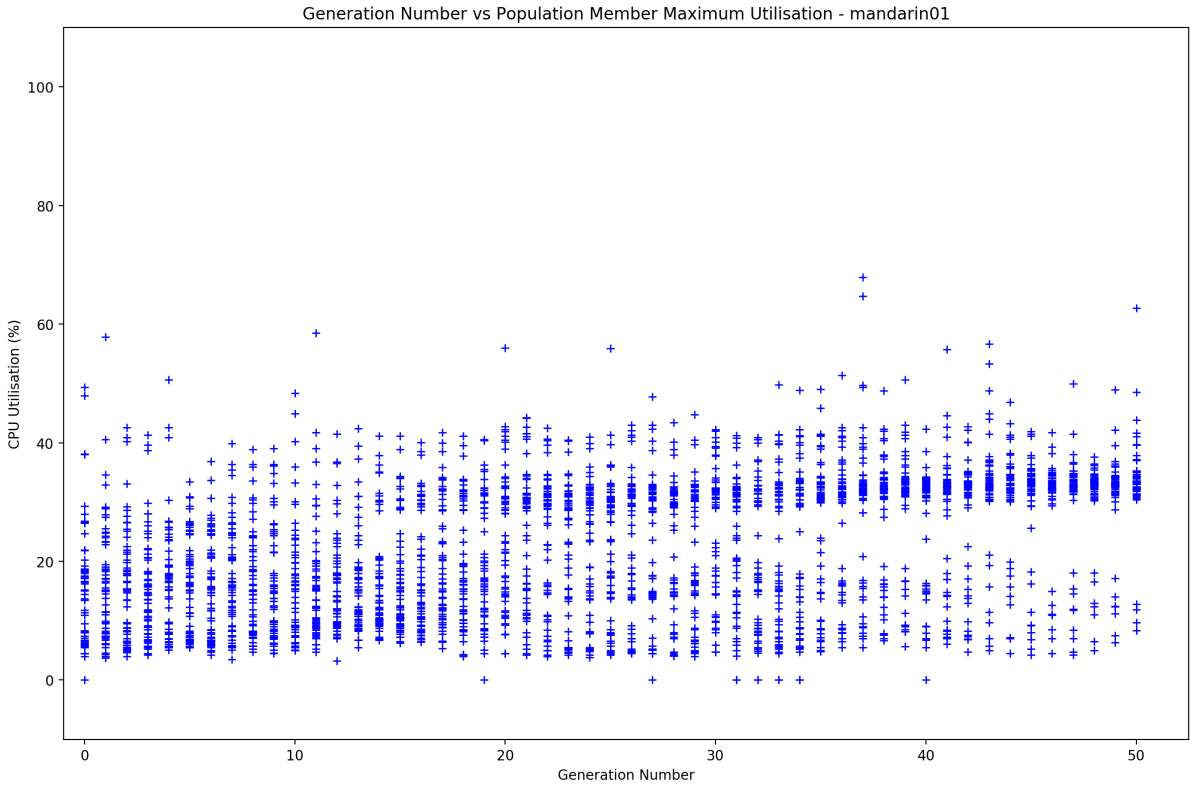

Figure E.9: Mandarin01 maximum CPU resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

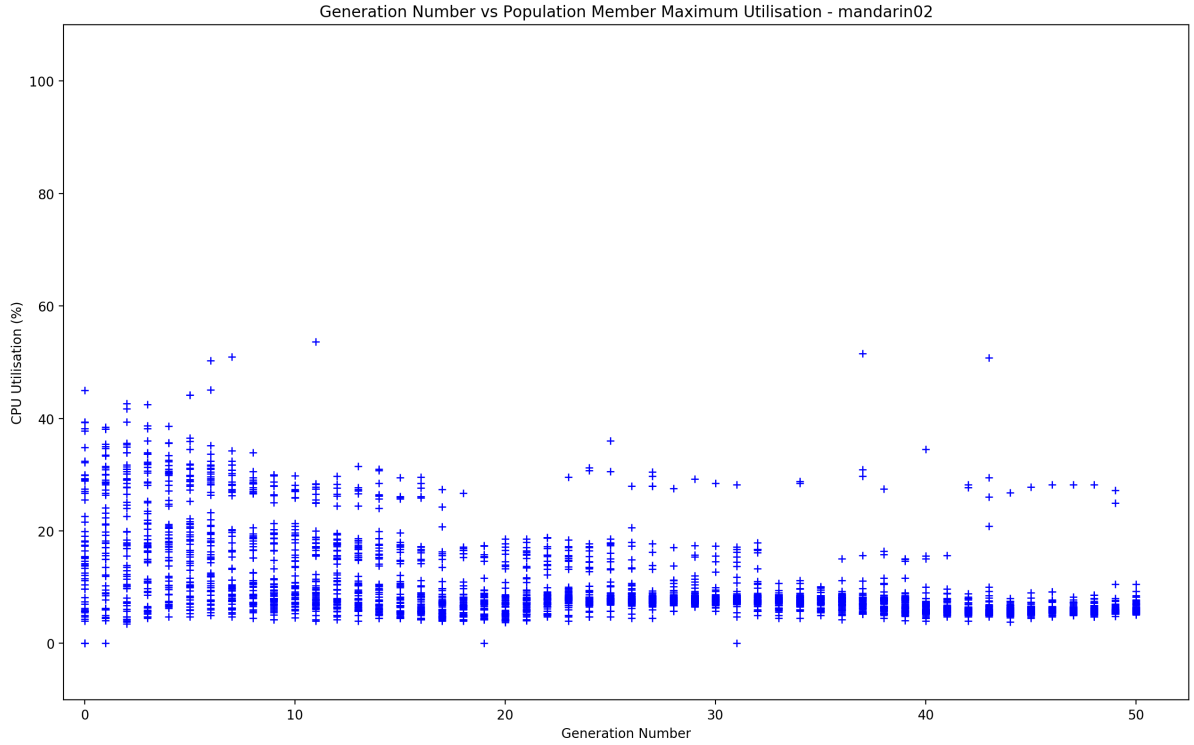

Figure E.10: Mandarin02 maximum CPU resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

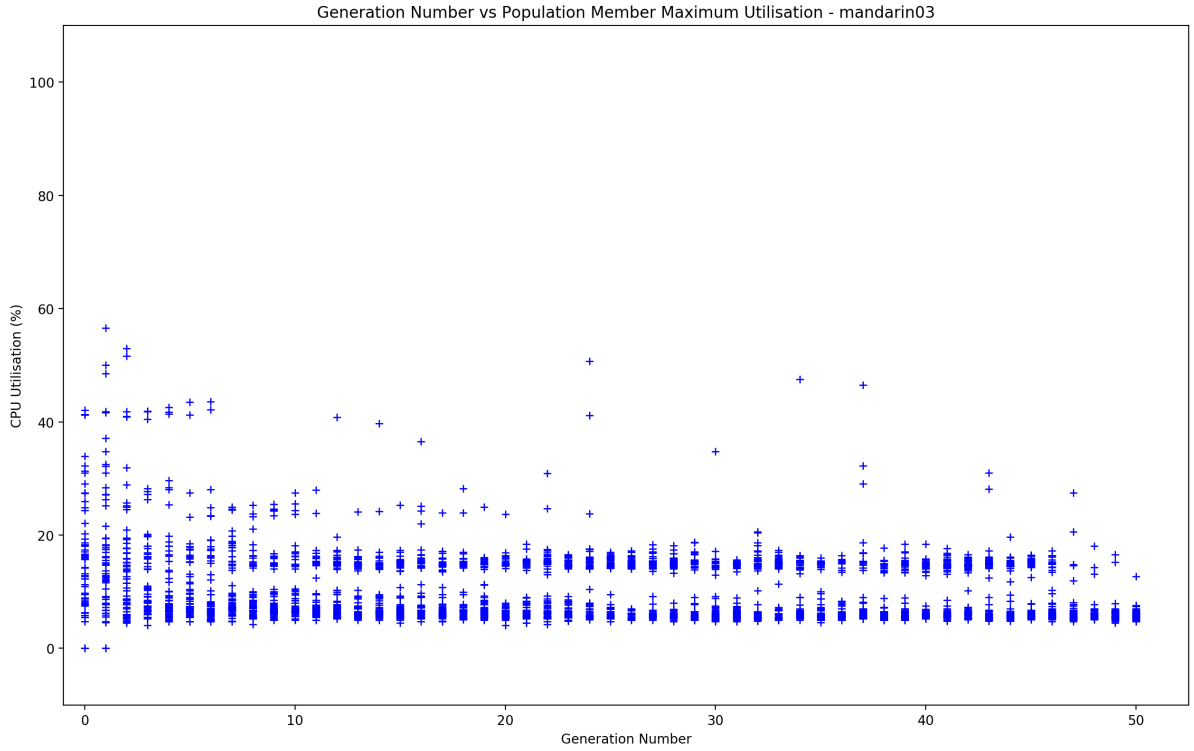

Figure E.11: Mandarin03 maximum CPU resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

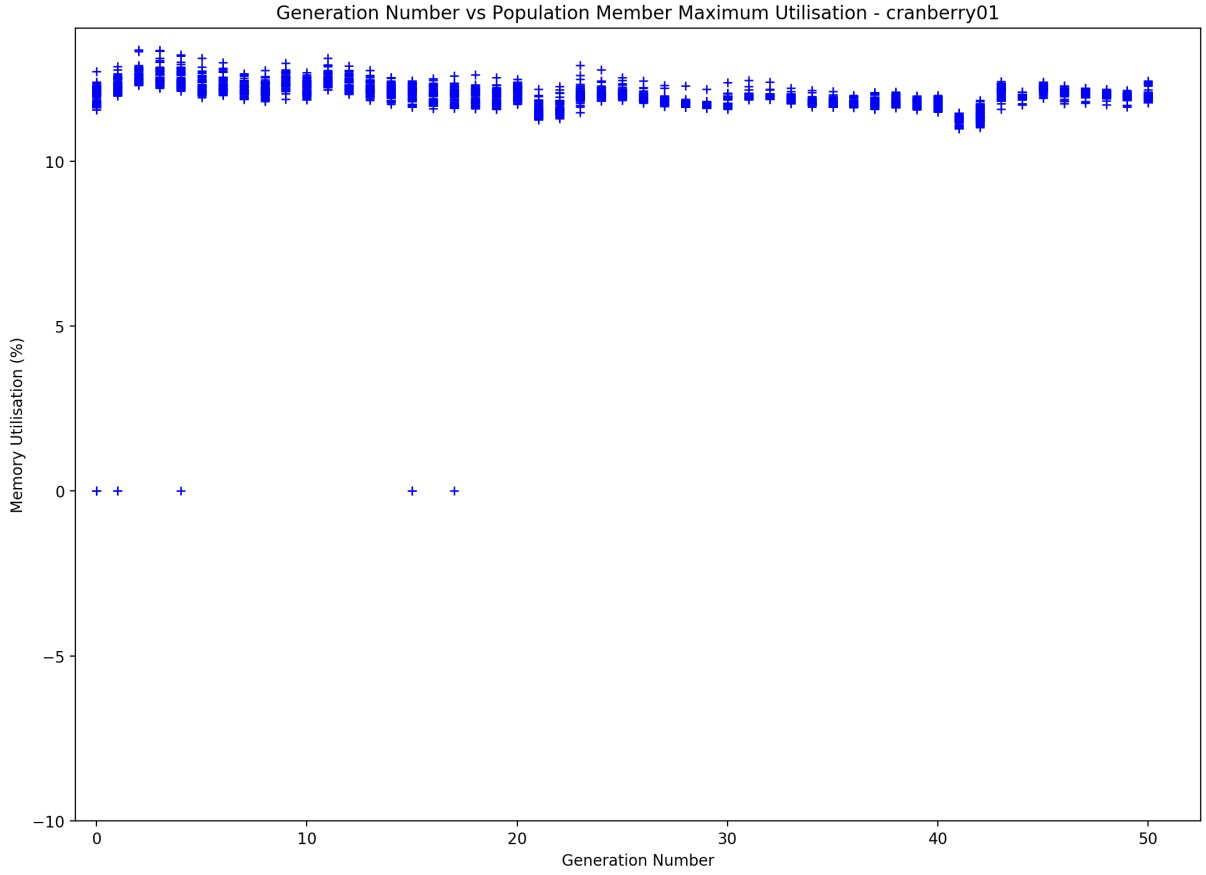

Figure E.12: Cranberry01 maximum memory resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

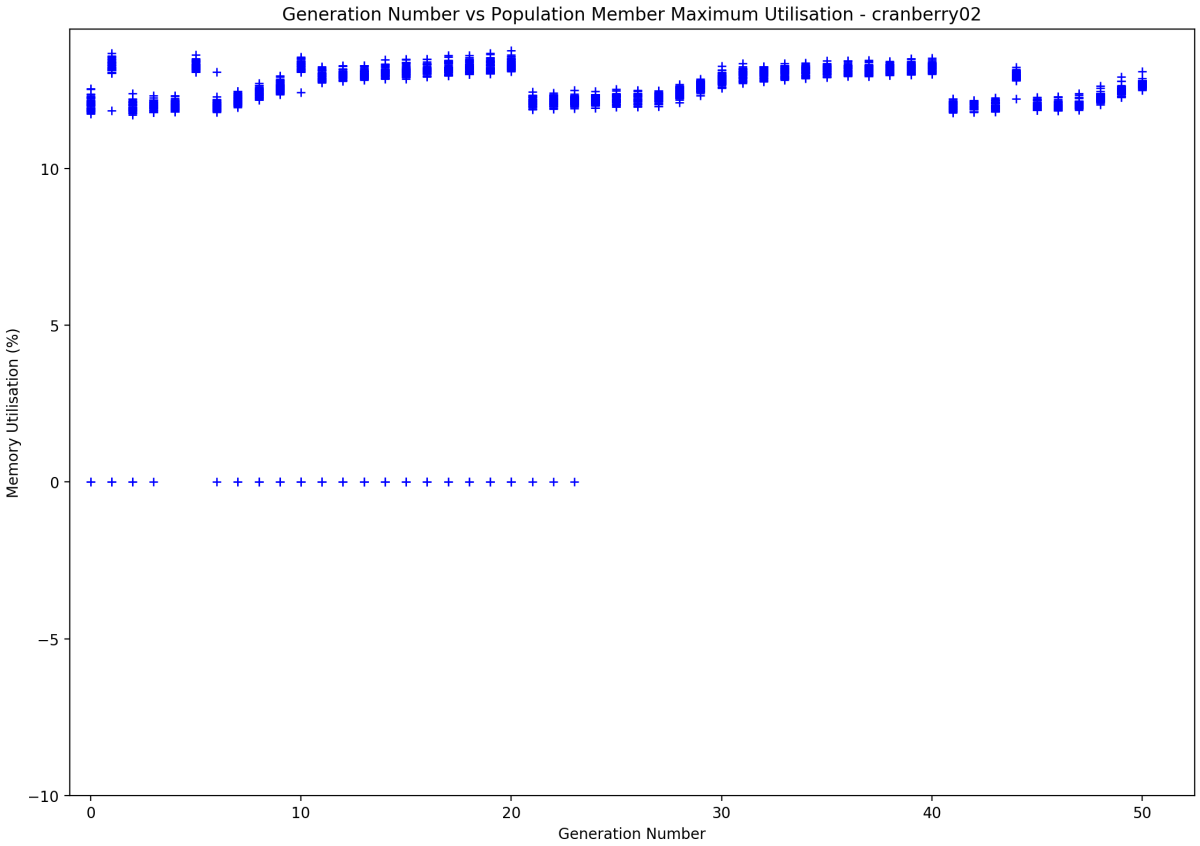

Figure E.13: Cranberry02 maximum memory resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

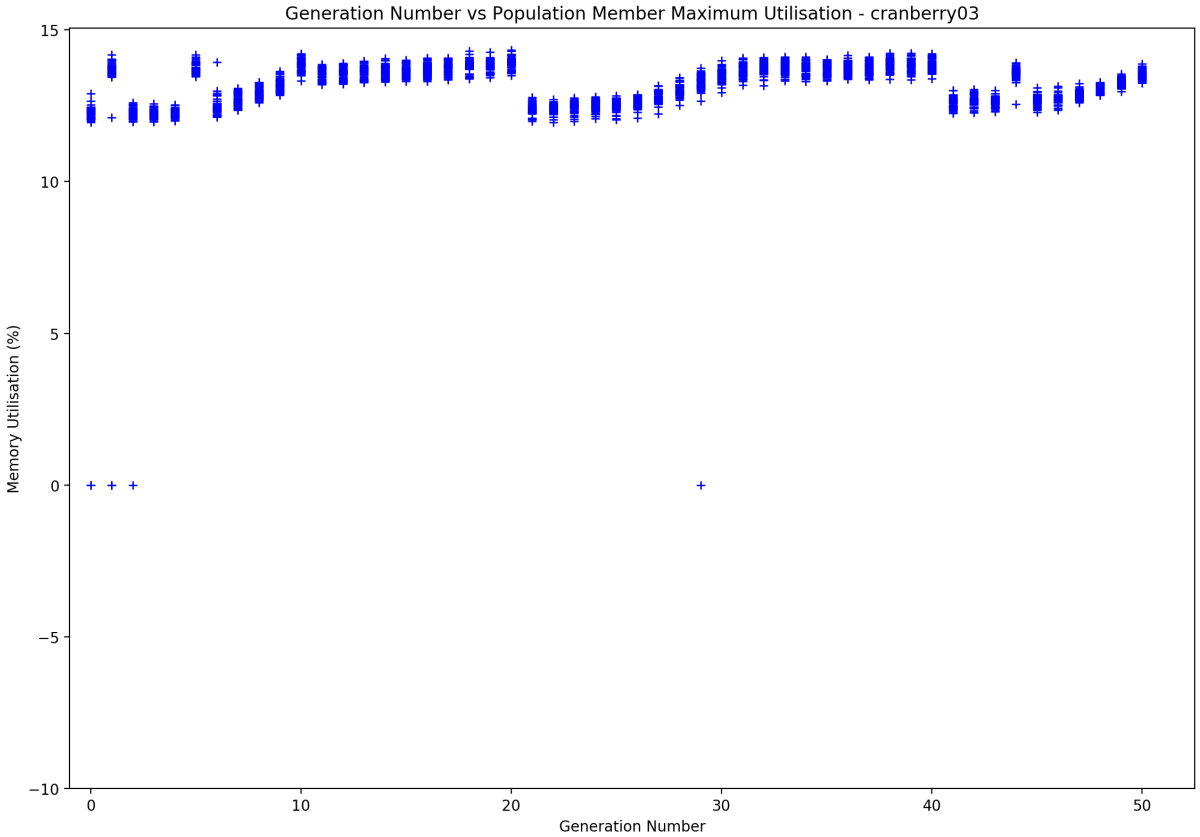

Figure E.14: Cranberry03 maximum memory resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

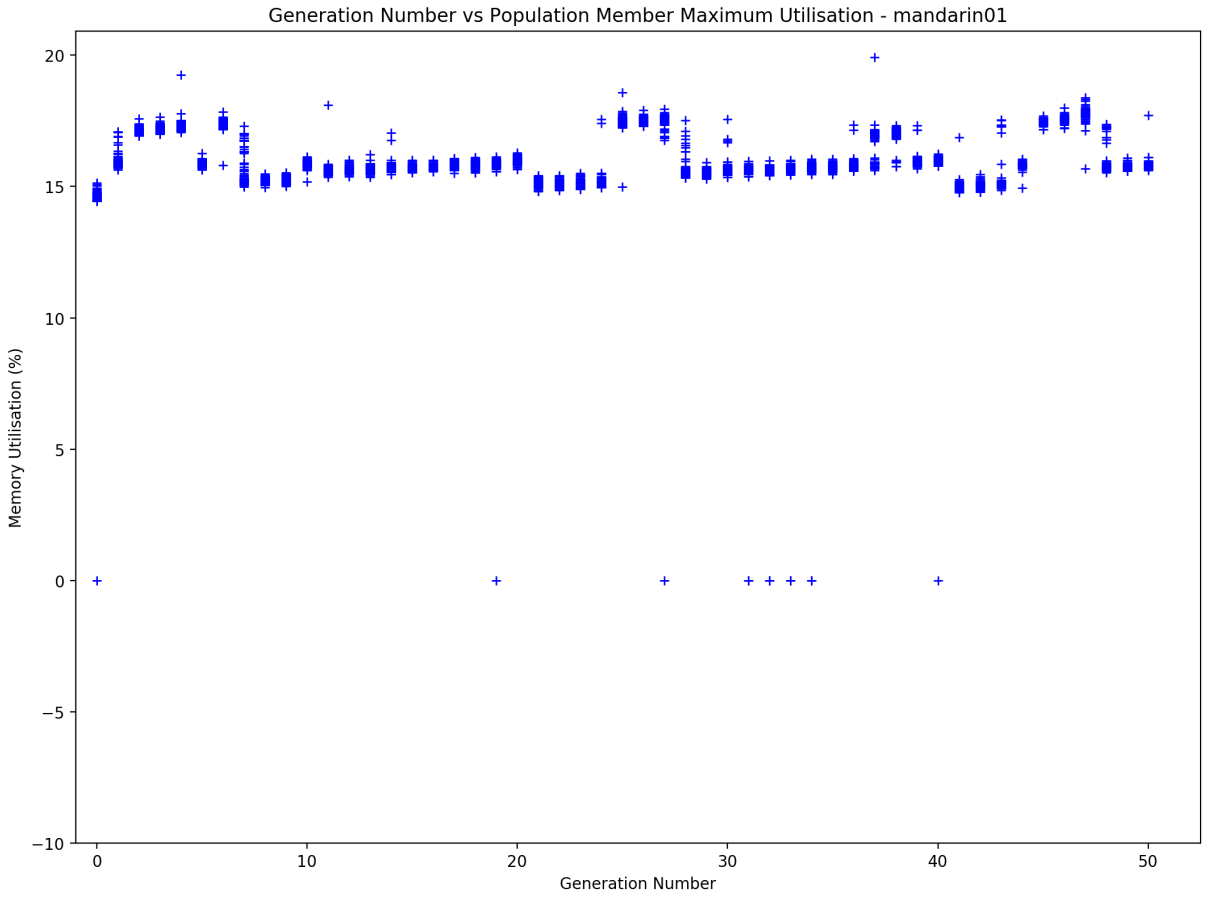

Figure E.15: Mandarin01 maximum memory resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

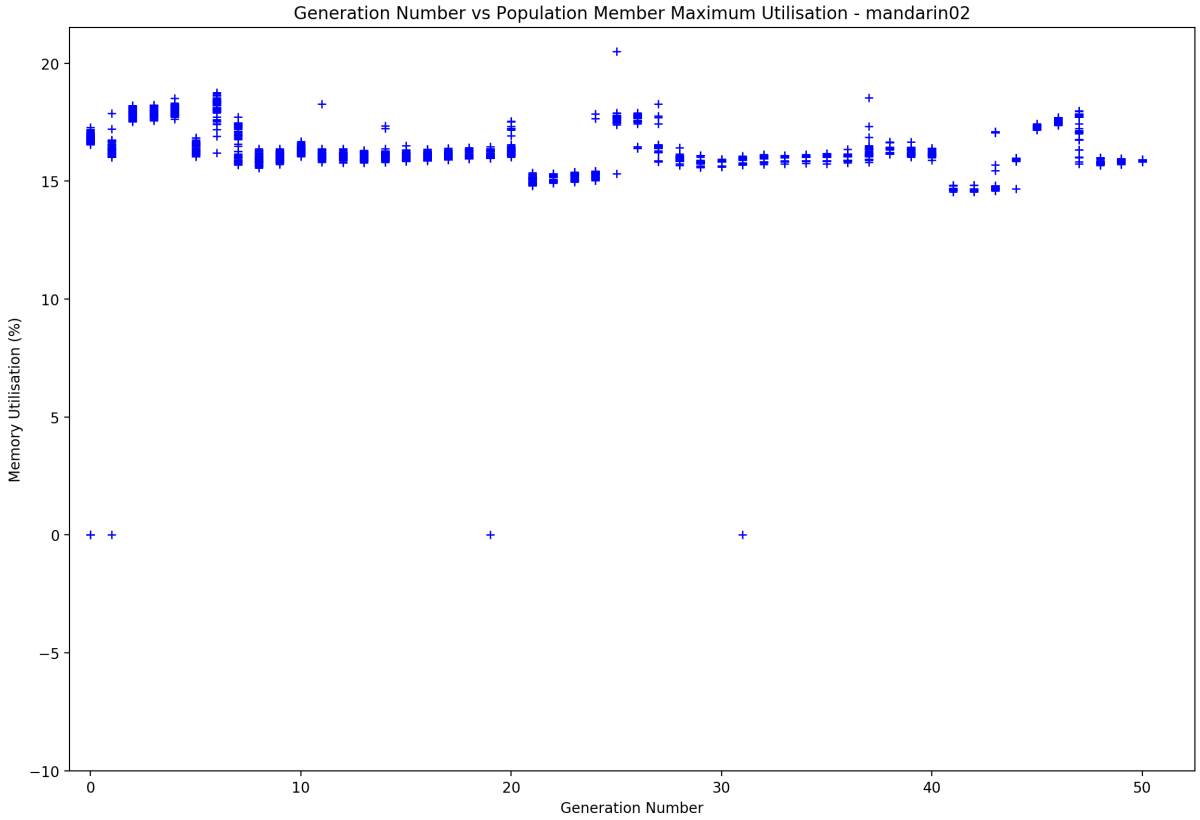

Figure E.16: Mandarin02 maximum memory resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

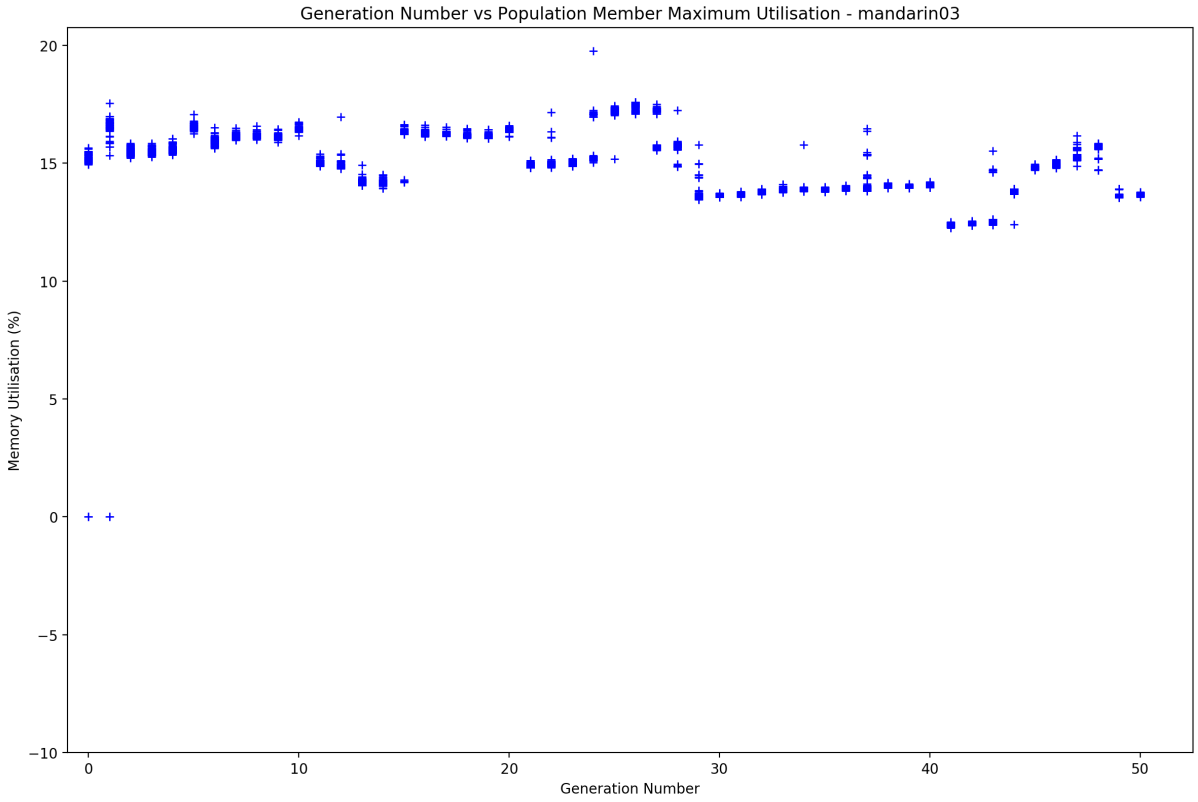

Figure E.17: Mandarin03 maximum memory resource consumption for an undersea sensor software system for the baseline experiment (50 generations with populations of 60, and 58 samples for each population member)

E.3 Case Study 0: Final Generation Array

Table E.1: Baseline case study final generation details

Population Number	Component Deployment Array	Objective Score
1993	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	349.534090099
2599	[1, 2, 4, 5, 2, 4, 0, 2, 2, 2, 3, 5, 2, 2, 3, 2, 2, 3, 3]	349.534090099
2713	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	349.534090099
2793	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	349.534090099
2833	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	349.534090099
2870	[1, 3, 4, 5, 3, 4, 0, 3, 2, 2, 2, 5, 2, 3, 2, 2, 2, 3, 3]	349.534090099
2912	[1, 2, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	349.534090099
2913	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	349.534090099
2918	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	349.534090099
2957	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 2, 5, 2, 3, 2, 2, 2, 3, 3]	349.534090099
2966	[1, 3, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 3, 2, 3, 2, 2, 3, 3]	349.534090099
2989	[1, 2, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	349.534090099
2999	[1, 2, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 3, 2, 3, 3, 2, 3, 3]	349.534090099
3006	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 2, 5, 2, 3, 2, 2, 2, 3, 3]	349.534090099
3021	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	349.534090099
3027	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 2, 3, 3, 2, 2, 3]	349.534090099
3032	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 2]	349.534090099
2195	[1, 0, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.113102898
2451	[1, 0, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.113102898
2518	[1, 3, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 2, 2, 3, 0, 2, 2, 3]	350.113102898
2596	[1, 2, 4, 5, 2, 4, 0, 2, 2, 0, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.113102898
2639	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 0, 5, 2, 3, 2, 2, 2, 2, 3]	350.113102898
2798	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 0, 3]	350.113102898
2823	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 0, 3]	350.113102898
2826	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 2, 3, 0, 2, 2, 3]	350.113102898
2874	[1, 2, 4, 5, 3, 4, 0, 0, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	350.113102898
2887	[1, 0, 4, 5, 2, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.113102898
2890	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 2, 3, 0, 2, 2, 3]	350.113102898
2891	[1, 2, 4, 5, 2, 4, 0, 2, 3, 2, 0, 5, 2, 3, 2, 2, 2, 2, 3]	350.113102898
2906	[1, 3, 4, 5, 2, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 0, 3]	350.113102898
2947	[1, 2, 4, 5, 3, 4, 0, 3, 2, 2, 0, 5, 2, 3, 2, 2, 2, 2, 3]	350.113102898

Continued on next page

Table E.1 – continued from previous page

Population Number	Component Deployment Array	Objective Score
2971	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 0, 5, 2, 3, 2, 2, 2, 2, 3]	350.113102898
3009	[1, 2, 4, 5, 2, 4, 0, 2, 3, 2, 0, 5, 3, 3, 2, 2, 2, 2, 3]	350.113102898
3039	[1, 0, 4, 5, 2, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.113102898
3040	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 0, 2, 2, 3]	350.113102898
3057	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 0, 5, 2, 3, 2, 2, 2, 2, 3]	350.113102898
2130	[1, 2, 4, 5, 3, 4, 0, 2, 1, 2, 3, 5, 2, 2, 3, 2, 2, 2, 3]	350.152309296
2476	[1, 3, 4, 5, 3, 4, 0, 2, 1, 3, 3, 5, 2, 2, 2, 2, 2, 2, 3]	350.152309296
2531	[1, 2, 4, 5, 3, 4, 0, 2, 1, 2, 3, 5, 2, 2, 3, 2, 2, 2, 3]	350.152309296
2543	[1, 3, 4, 5, 3, 4, 0, 2, 1, 3, 3, 5, 2, 2, 2, 2, 2, 2, 3]	350.152309296
2568	[1, 1, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	350.152309296
2573	[1, 1, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	350.152309296
2574	[1, 2, 4, 5, 3, 4, 0, 2, 1, 2, 3, 5, 2, 2, 3, 2, 2, 2, 3]	350.152309296
2677	[1, 2, 4, 5, 3, 4, 0, 2, 3, 2, 3, 5, 2, 2, 2, 2, 3, 1, 2]	350.152309296
2736	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 1, 5, 2, 2, 3, 2, 2, 2, 3]	350.152309296
2761	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 1, 3, 3, 2, 2, 2, 3]	350.152309296
2768	[1, 2, 4, 5, 3, 4, 0, 2, 1, 3, 3, 5, 2, 3, 2, 2, 3, 2, 3]	350.152309296
2841	[1, 2, 4, 5, 3, 4, 0, 2, 1, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.152309296
2864	[1, 3, 4, 5, 3, 4, 0, 2, 1, 3, 3, 5, 2, 2, 3, 2, 2, 3, 3]	350.152309296
2892	[1, 1, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 2, 3, 2, 2, 3, 3]	350.152309296
2904	[1, 3, 4, 5, 3, 4, 0, 2, 1, 2, 3, 5, 2, 2, 3, 2, 2, 3, 3]	350.152309296
2915	[1, 1, 4, 5, 3, 4, 0, 2, 2, 2, 2, 5, 2, 3, 2, 2, 2, 3, 3]	350.152309296
2928	[1, 3, 4, 5, 3, 4, 0, 2, 1, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.152309296
2940	[1, 2, 4, 5, 3, 4, 0, 1, 2, 3, 3, 5, 2, 2, 3, 2, 2, 2, 3]	350.152309296
2980	[1, 3, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 2, 3, 2, 2, 2, 1, 3]	350.152309296
3004	[1, 3, 4, 5, 3, 4, 0, 2, 1, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	350.152309296
3012	[1, 2, 4, 5, 3, 4, 0, 1, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.152309296
3026	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 2, 2, 2, 2, 1, 2]	350.152309296
2946	[0, 1, 4, 5, 3, 4, 3, 1, 1, 1, 1, 5, 2, 2, 2, 2, 2, 1, 2]	350.207174541
2516	[1, 3, 4, 5, 0, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.402609297

Table E.2: Baseline case study unique deployment options

Number	Unique Component Deployment Array	Objective Score
1	[0, 1, 4, 5, 3, 4, 3, 1, 1, 1, 1, 5, 2, 2, 2, 2, 2, 1, 2]	350.20
2	[1, 0, 4, 5, 2, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.11
3	[1, 2, 4, 5, 3, 4, 0, 0, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	350.11
4	[1, 0, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.11
5	[1, 1, 4, 5, 3, 4, 0, 2, 2, 2, 2, 5, 2, 3, 2, 2, 2, 3, 3]	350.15
6	[1, 1, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 2, 3, 2, 2, 3, 3]	350.15
7	[1, 1, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	350.15
8	[1, 2, 4, 5, 2, 4, 0, 2, 2, 0, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.11
9	[1, 2, 4, 5, 2, 4, 0, 2, 2, 2, 3, 5, 2, 2, 3, 2, 2, 3, 3]	349.53
10	[1, 2, 4, 5, 2, 4, 0, 2, 3, 2, 0, 5, 2, 3, 2, 2, 2, 2, 3]	349.53
11	[1, 2, 4, 5, 3, 4, 0, 1, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.15
12	[1, 2, 4, 5, 3, 4, 0, 1, 2, 3, 3, 5, 2, 2, 3, 2, 2, 2, 3]	350.15
13	[1, 2, 4, 5, 3, 4, 0, 2, 1, 2, 3, 5, 2, 2, 3, 2, 2, 2, 3]	350.15
14	[1, 2, 4, 5, 3, 4, 0, 2, 1, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.15
15	[1, 2, 4, 5, 3, 4, 0, 2, 1, 3, 3, 5, 2, 3, 2, 2, 3, 2, 3]	350.15
16	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 0, 5, 2, 3, 2, 2, 2, 2, 3]	349.53
17	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 1, 5, 2, 2, 3, 2, 2, 2, 3]	350.15
18	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 2, 5, 2, 3, 2, 2, 2, 3, 3]	350.15
19	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 1, 3, 3, 2, 2, 2, 3]	350.15
20	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 2, 3, 0, 2, 2, 3]	350.11
21	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 2, 3, 3, 2, 2, 3]	350.11
22	[1, 2, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 0, 3]	350.11
23	[1, 2, 4, 5, 3, 4, 0, 3, 2, 2, 0, 5, 2, 3, 2, 2, 2, 2, 3] 350.11	
24	[1, 2, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	349.53
25	[1, 2, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 3, 2, 3, 3, 2, 3, 3]	349.53
26	[1, 3, 4, 5, 0, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.40
27	[1, 3, 4, 5, 2, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 0, 3]	350.11
28	[1, 3, 4, 5, 3, 4, 0, 2, 1, 2, 3, 5, 2, 2, 3, 2, 2, 3, 3]	350.15
29	[1, 3, 4, 5, 3, 4, 0, 2, 1, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.15
30	[1, 3, 4, 5, 3, 4, 0, 2, 1, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	350.15
31	[1, 3, 4, 5, 3, 4, 0, 2, 1, 3, 3, 5, 2, 2, 2, 2, 2, 2, 3]	350.15
32	[1, 3, 4, 5, 3, 4, 0, 2, 1, 3, 3, 5, 2, 2, 3, 2, 2, 3, 3]	350.15
33	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 2, 2, 2, 2, 1, 2]	350.15
34	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 0, 2, 2, 3]	350.11

Continued on next page

Table E.2 – continued from previous page

Number	Unique Component Deployment Array	Objective Score
35	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 2]	349.53
36	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 2, 2, 2, 2, 3]	350.15
37	[1, 3, 4, 5, 3, 4, 0, 2, 2, 2, 3, 5, 2, 3, 3, 2, 2, 2, 3]	349.53
38	[1, 3, 4, 5, 3, 4, 0, 3, 2, 2, 2, 5, 2, 3, 2, 2, 2, 3, 3]	349.53
39	[1, 3, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 2, 2, 3, 0, 2, 2, 3]	350.11
40	[1, 3, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 2, 3, 2, 2, 2, 1, 3]	350.15
41	[1, 3, 4, 5, 3, 4, 0, 3, 2, 2, 3, 5, 3, 2, 3, 2, 2, 3, 3]	349.53

E.4 Case Study 1: Final Generation Array

Table E.3: Case study 1 final generation details

Number	Component Deployment Array	Objective Score
875	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
910	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
930	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
962	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1053	[0, 2, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1058	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.604161459
1083	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1090	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1145	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.604161459
1150	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1209	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.604161459
1223	[0, 2, 4, 0, 2, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.604161459
1234	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 2, 4, 4, 2, 1, 1, 1, 2, 2]	358.604161459
1241	[0, 2, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.604161459
1259	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.604161459
1274	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1275	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 2, 4, 4, 1, 2, 1, 1, 1, 2]	358.604161459
1277	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.604161459

Continued on next page

Table E.3 – continued from previous page

Number	Component Deployment Array	Objective Score
1330	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.604161459
1348	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1365	[0, 2, 4, 0, 2, 3, 5, 1, 1, 1, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.604161459
1375	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 2, 2]	358.604161459
1389	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.604161459
1391	[0, 2, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.604161459
1401	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1406	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.604161459
1417	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1429	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 2, 2]	358.604161459
1430	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1433	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.604161459
1444	[0, 2, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.604161459
1452	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.604161459
1453	[0, 2, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1454	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.604161459
1457	[0, 1, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1460	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.604161459
1461	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1462	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.604161459
1484	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 2, 4, 4, 1, 2, 1, 1, 1, 1]	358.604161459
1489	[0, 1, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 2, 2, 1, 1, 1]	358.604161459
1526	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.604161459
1542	[0, 1, 4, 0, 1, 3, 5, 2, 2, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.604161459
1568	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.604161459
1575	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.604161459
1582	[0, 2, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.604161459
1584	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1585	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.604161459
1591	[0, 1, 4, 0, 1, 3, 5, 2, 2, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.604161459
1604	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.604161459
1609	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 2, 1]	358.604161459
1614	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.604161459
1622	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.604161459
1623	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.604161459

Continued on next page

Table E.3 – continued from previous page

Number	Component Deployment Array	Objective Score
1629	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 2, 4, 4, 2, 1, 2, 1, 1, 1]	358.604161459
1631	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.604161459
1636	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.604161459
1637	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1644	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459
1646	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.604161459
1647	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.604161459

Table E.4: Case study 1 unique deployment options

Number	Component Deployment Array	Objective Score
1	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
2	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
3	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.60
4	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.60
5	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.60
6	[0, 1, 4, 0, 1, 3, 5, 1, 1, 2, 2, 4, 4, 1, 2, 1, 1, 1, 2]	358.60
7	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
8	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 2, 2]	358.60
9	[0, 1, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
10	[0, 1, 4, 0, 1, 3, 5, 2, 2, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.60
11	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
12	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.60
13	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 2, 4, 4, 1, 2, 1, 1, 1, 1]	358.60
14	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 2, 4, 4, 2, 1, 1, 1, 2, 2]	358.60
15	[0, 1, 4, 0, 2, 3, 5, 1, 1, 2, 2, 4, 4, 2, 1, 2, 1, 1, 1]	358.60
16	[0, 1, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
17	[0, 1, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 2, 2, 1, 1, 1]	358.60
18	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
19	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 2, 1]	358.60
20	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
21	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.60
22	[0, 1, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.60

Continued on next page

Table E.4 – continued from previous page

Number	Component Deployment Array	Objective Score
23	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
24	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.60
25	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.60
26	[0, 1, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 2, 2, 1, 1, 1, 1]	358.60
27	[0, 2, 4, 0, 1, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
28	[0, 2, 4, 0, 2, 3, 5, 1, 1, 1, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
29	[0, 2, 4, 0, 2, 3, 5, 1, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
30	[0, 2, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
31	[0, 2, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
32	[0, 2, 4, 0, 2, 3, 5, 2, 1, 1, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.60
33	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60
34	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 1, 2, 1, 1, 1]	358.60
35	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 1]	358.60
36	[0, 2, 4, 0, 2, 3, 5, 2, 1, 2, 1, 4, 4, 1, 2, 1, 1, 1, 2]	358.60
37	[0, 2, 4, 0, 2, 3, 5, 2, 2, 2, 1, 4, 4, 1, 1, 1, 1, 1, 1]	358.60

E.5 Case Study 2: Final Generation Array

Table E.5: Case study 2 final generation details

Population Number	Component Deployment Array	Objective Score
2394	[4, 4, 0, 3, 4, 1, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	304.749652395
2518	[4, 4, 0, 3, 4, 1, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	304.749652395
2814	[4, 4, 0, 3, 4, 1, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	304.749652395
2949	[4, 4, 0, 3, 4, 1, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	304.749652395
2959	[4, 4, 0, 3, 4, 1, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	304.749652395
2376	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2385	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2407	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2502	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2536	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441

Continued on next page

Table E.5 – continued from previous page

Population Number	Component Deployment Array	Objective Score
2679	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2706	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2738	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2766	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2793	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2807	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2829	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2895	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2896	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2925	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2934	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2980	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2992	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
3030	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
3044	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.778902441
2038	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2163	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2168	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2188	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2225	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2261	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2311	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2320	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2339	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2359	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2413	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2419	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2441	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2495	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2497	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2546	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2620	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2623	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2661	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2667	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276

Continued on next page

Table E.5 – continued from previous page

Population Number	Component Deployment Array	Objective Score
2731	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2771	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2856	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2885	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2909	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2915	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2921	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2932	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2941	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2942	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2951	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
2982	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
3004	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
3014	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276
3048	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.274040276

Table E.6: Case study 2 final generation unique deployment options details

Number	Component Deployment Array	Objective Score
1	[4, 4, 0, 3, 4, 1, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	304.75
2	[4, 4, 0, 2, 4, 0, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	304.78
3	[4, 4, 0, 2, 4, 1, 5, 4, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	305.27

E.6 Case Study 3: Final Generation Array

Table E.7: Case study 3 final generation details

Population Number	Component Deployment Array	Objective Score
2997	[1, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	769.347629203
2788	[2, 4, 3, 1, 4, 0, 5, 4, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]	774.892021322
2900	[2, 4, 1, 0, 4, 3, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	781.933458614
2754	[2, 4, 3, 1, 4, 0, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	784.500950446
2370	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	788.378725669
2702	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	788.378725669
2863	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	788.378725669
2937	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	788.378725669
2960	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	788.378725669
3016	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	788.378725669
3060	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]	791.274167565
2904	[2, 4, 1, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]	791.449641165
3023	[2, 4, 1, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]	791.449641165
2599	[1, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 1, 4]	791.660820973
2474	[2, 4, 3, 0, 4, 2, 5, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4]	801.553095173
3012	[2, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]	801.553095173
3015	[2, 4, 3, 0, 4, 2, 5, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4]	801.553095173
2261	[2, 4, 3, 1, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	803.709175616
2872	[2, 4, 3, 1, 4, 0, 5, 4, 0, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	805.211913644
2886	[2, 4, 3, 1, 4, 0, 5, 4, 4, 4, 4, 4, 4, 4, 0, 4, 4, 0, 4]	805.211913644
2901	[2, 4, 3, 1, 4, 0, 5, 4, 0, 4, 4, 4, 4, 4, 0, 4, 4, 4, 4]	805.211913644
2564	[2, 4, 3, 0, 4, 1, 5, 4, 4, 1, 4, 4, 4, 4, 4, 1, 4, 4, 4]	810.342599649
3045	[2, 4, 3, 0, 4, 1, 5, 4, 4, 1, 4, 4, 4, 4, 4, 1, 4, 4, 4]	810.342599649
2209	[1, 4, 3, 0, 4, 2, 5, 4, 0, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	810.439910649
2593	[2, 4, 1, 0, 4, 2, 5, 3, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	811.146775119
2735	[2, 4, 3, 1, 4, 2, 5, 4, 4, 4, 4, 0, 4, 4, 4, 0, 4, 4, 4]	812.36729737
2815	[2, 4, 3, 1, 4, 2, 5, 4, 4, 4, 4, 0, 4, 4, 4, 0, 4, 4, 4]	812.36729737
2945	[2, 4, 3, 1, 4, 2, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 0, 4]	812.36729737
2993	[2, 4, 3, 1, 4, 2, 5, 4, 4, 4, 4, 0, 4, 4, 4, 0, 4, 4, 4]	812.36729737
2850	[2, 4, 3, 1, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	813.039196726
2150	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	813.858654536
2237	[2, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	813.858654536
2288	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	813.858654536
2384	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	813.858654536

Continued on next page

Table E.7 – continued from previous page

Population Number	Component Deployment Array	Objective Score
2472	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	813.858654536
2513	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	813.858654536
2784	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	813.858654536
2877	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	813.858654536
2953	[2, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	813.858654536
2964	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 1, 4]	813.858654536
3029	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	813.858654536
3035	[2, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	813.858654536
1617	[1, 4, 3, 0, 4, 2, 5, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 0, 4]	815.899427159
2191	[2, 4, 3, 0, 4, 1, 5, 4, 0, 4, 4, 4, 4, 4, 1, 4, 4, 4, 4]	815.899427159
2917	[1, 0, 3, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 4]	815.899427159
2284	[0, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	816.273333477
2382	[0, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	816.273333477
2389	[0, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	816.273333477
2469	[0, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	816.273333477
2594	[0, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	816.273333477
2769	[0, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	816.273333477
2773	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	817.271034516
2914	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	817.271034516
2982	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	817.271034516
2346	[2, 4, 1, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	817.406878099
2698	[2, 4, 1, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	817.406878099
2911	[2, 4, 1, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	817.406878099
3041	[2, 4, 1, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	817.406878099
2962	[2, 4, 3, 1, 4, 0, 5, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	817.490440818
1996	[0, 4, 3, 1, 4, 2, 5, 4, 0, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	819.414913421

Table E.8: Case study 3 final generation unique component deployment details

Number	Component Deployment Array	Objective Score
1	[1, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	769.35
2	[2, 4, 3, 1, 4, 0, 5, 4, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]	774.89
3	[2, 4, 1, 0, 4, 3, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	781.93

Continued on next page

Table E.8 – continued from previous page

Number	Component Deployment Array	Objective Score
4	[2, 4, 3, 1, 4, 0, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	784.50
5	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	788.38
6	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]	791.27
7	[2, 4, 1, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]	791.45
8	[1, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 1, 4]	791.67
9	[2, 4, 3, 0, 4, 2, 5, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4]	801.55
10	[2, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]	801.55
11	[2, 4, 3, 1, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	803.71
12	[2, 4, 3, 1, 4, 0, 5, 4, 0, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	805.21
13	[2, 4, 3, 1, 4, 0, 5, 4, 0, 4, 4, 4, 4, 4, 0, 4, 4, 4, 4]	805.21
14	[2, 4, 3, 1, 4, 0, 5, 4, 4, 4, 4, 4, 4, 4, 0, 4, 4, 0, 4]	805.21
15	[2, 4, 3, 0, 4, 1, 5, 4, 4, 1, 4, 4, 4, 4, 4, 1, 4, 4, 4]	810.34
16	[1, 4, 3, 0, 4, 2, 5, 4, 0, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4]	810.44
17	[2, 4, 1, 0, 4, 2, 5, 3, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4]	811.15
18	[2, 4, 3, 1, 4, 2, 5, 4, 4, 4, 4, 0, 4, 4, 4, 0, 4, 4, 4]	812.37
19	[2, 4, 3, 1, 4, 2, 5, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 0, 4]	812.37
20	[2, 4, 3, 1, 4, 2, 5, 4, 4, 4, 4, 0, 4, 4, 4, 0, 4, 4, 4]	812.37
21	[2, 4, 3, 1, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	813.04
22	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	813.86
23	[2, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	813.86
24	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	813.86
25	[2, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	813.86
26	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 1, 4]	813.86
27	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	813.86
28	[2, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	813.86
29	[1, 4, 3, 0, 4, 2, 5, 4, 4, 4, 2, 4, 4, 4, 4, 4, 4, 0, 4]	815.90
30	[2, 4, 3, 0, 4, 1, 5, 4, 0, 4, 4, 4, 4, 4, 1, 4, 4, 4, 4]	815.90
31	[1, 0, 3, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 2, 4, 4, 4, 4]	815.90
32	[0, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	816.27
33	[0, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 1, 4, 4, 4]	816.27
34	[0, 1, 3, 0, 4, 2, 5, 4, 4, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	816.27
35	[2, 4, 3, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	817.27
36	[2, 4, 1, 0, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	817.41
37	[2, 4, 3, 1, 4, 0, 5, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4]	817.50
38	[0, 4, 3, 1, 4, 2, 5, 4, 0, 4, 4, 1, 4, 4, 4, 4, 4, 4, 4]	819.41

F. CEML DSML UML Class Diagrams

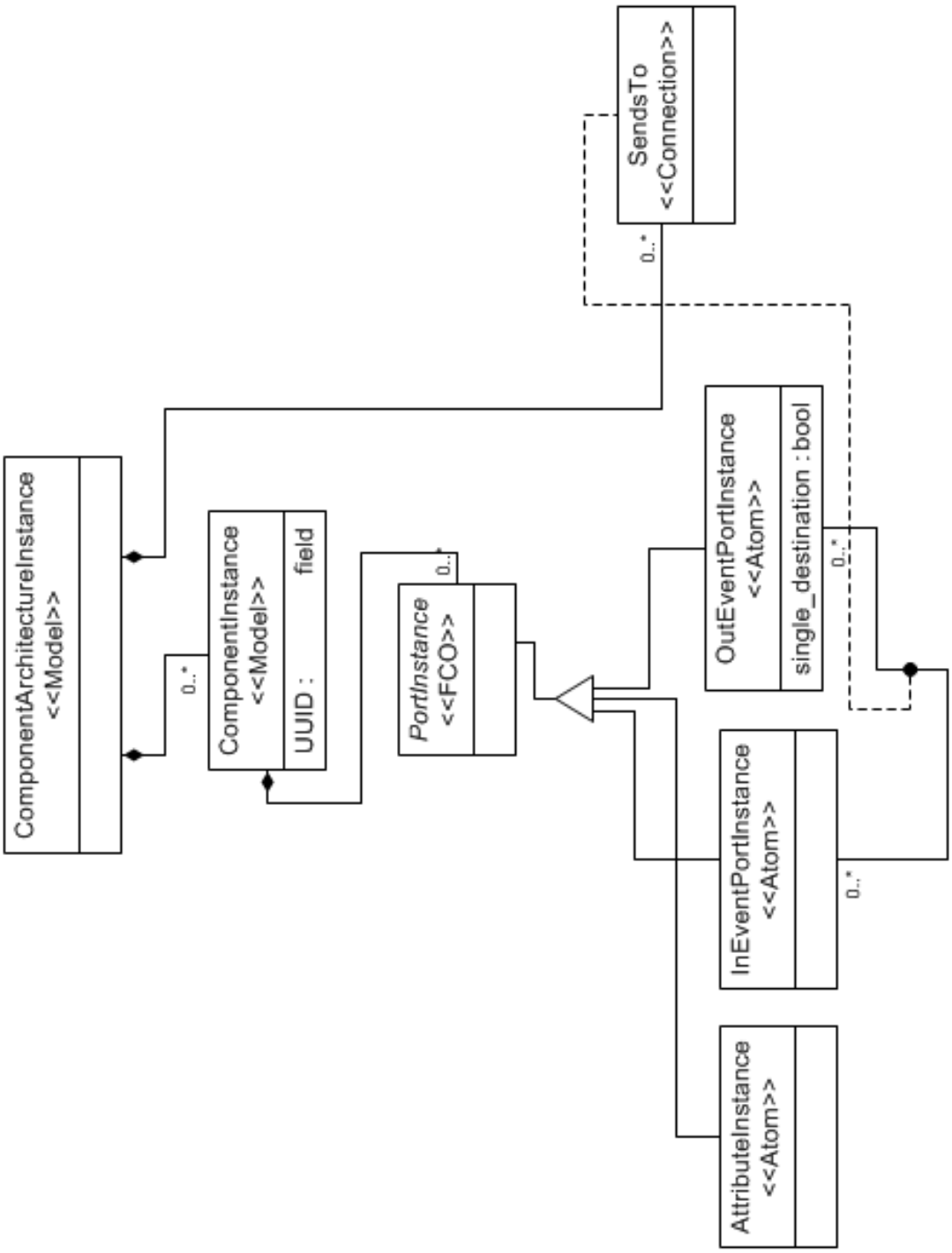

Figure F.1: CEML DSML component architecture modelling UML class diagram

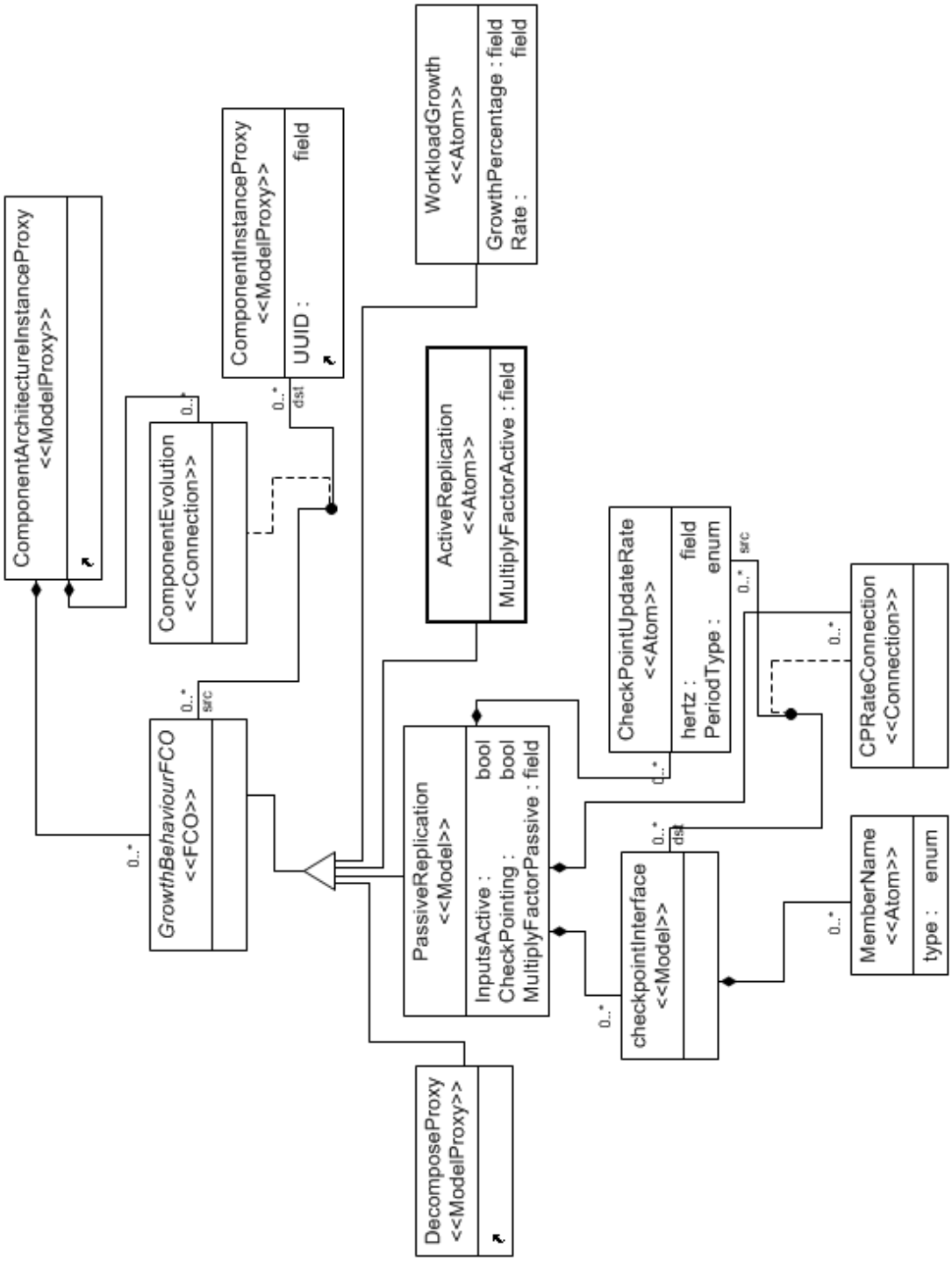

Figure F.2: CEML DSML component evolution modelling UML class diagram

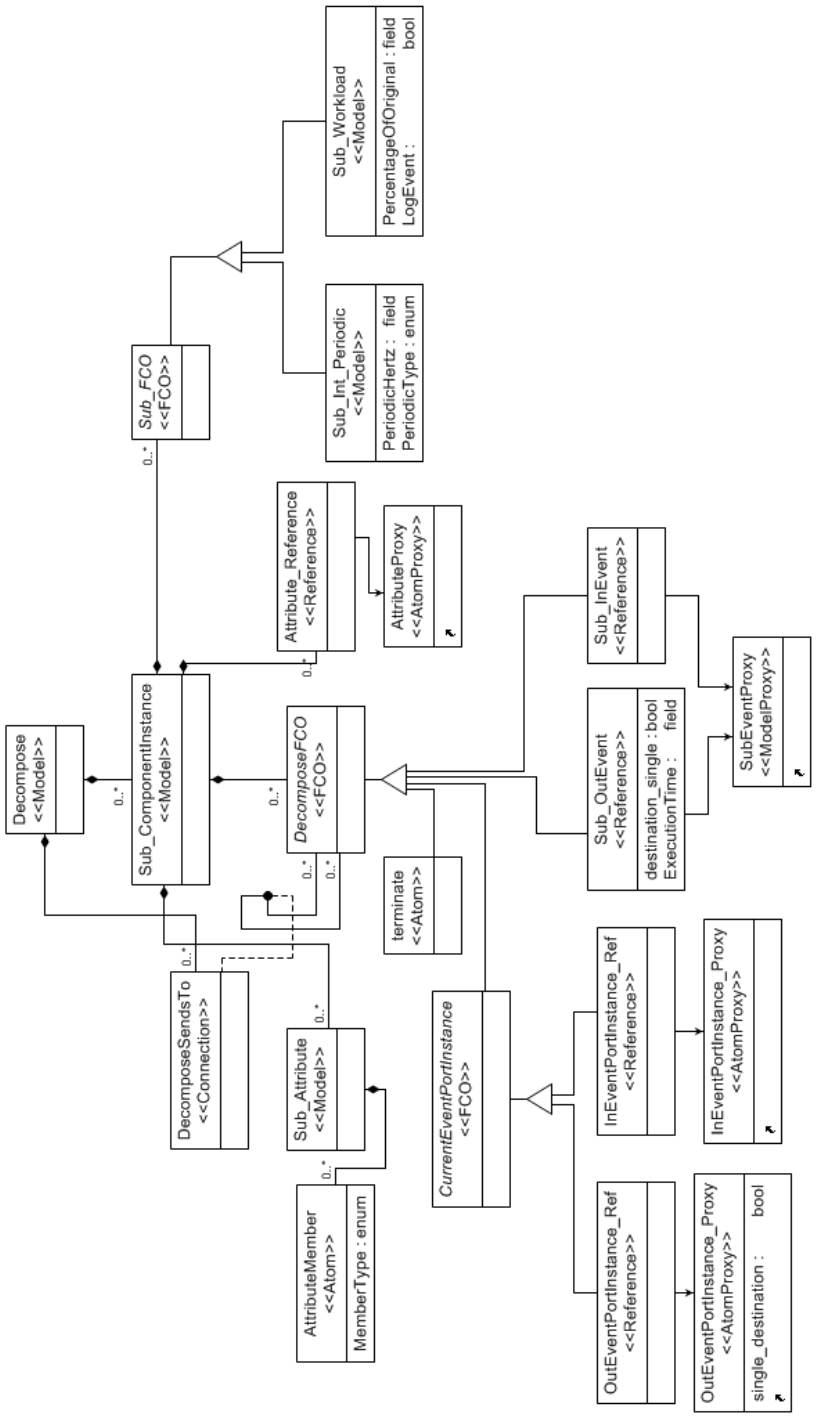

Figure F.3: CEML DSML component decomposing modelling UML class diagram

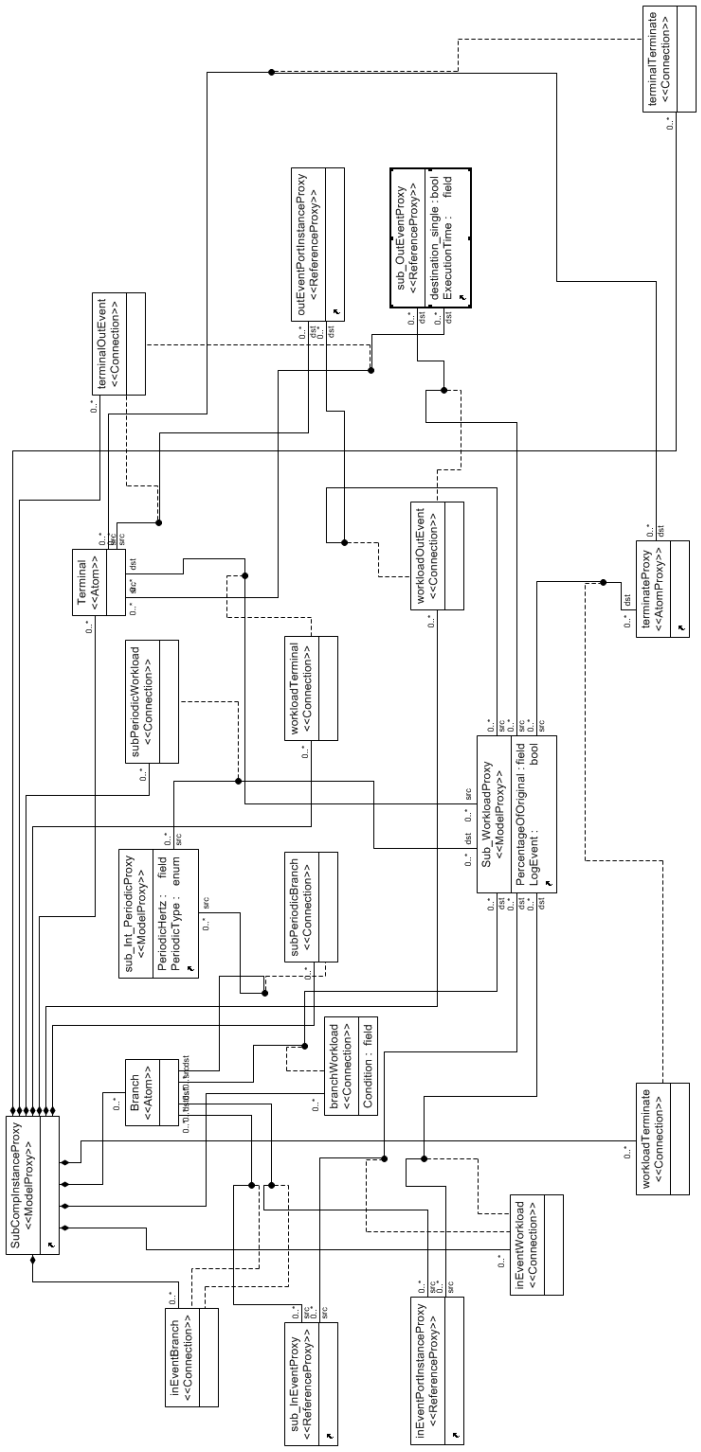

Figure F.4: CEML DSML component event modelling UML class diagram

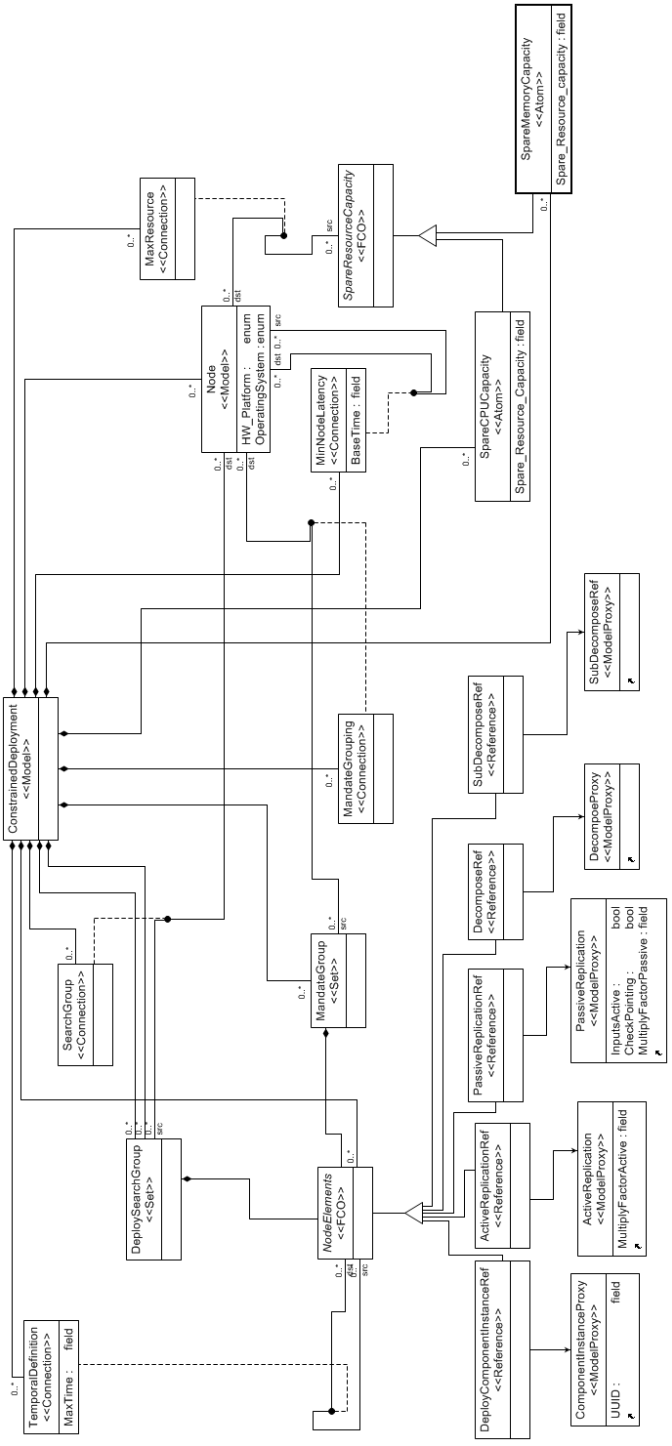

Figure F.5: CEML DSML component deployment constraint modelling UML class diagram

G. CEML Case Study Model Diagrams

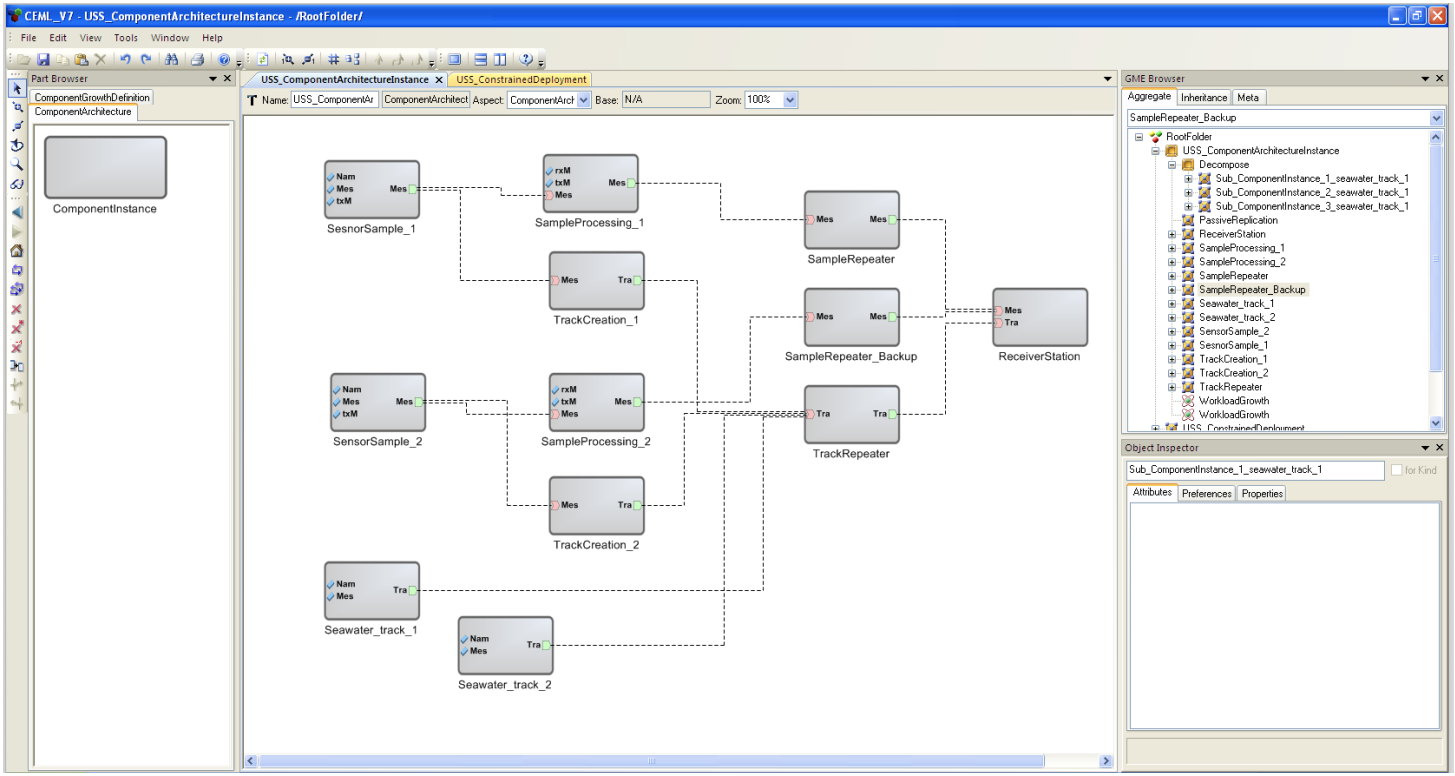

Figure G.1: Component architecture within the CEML modelling environment

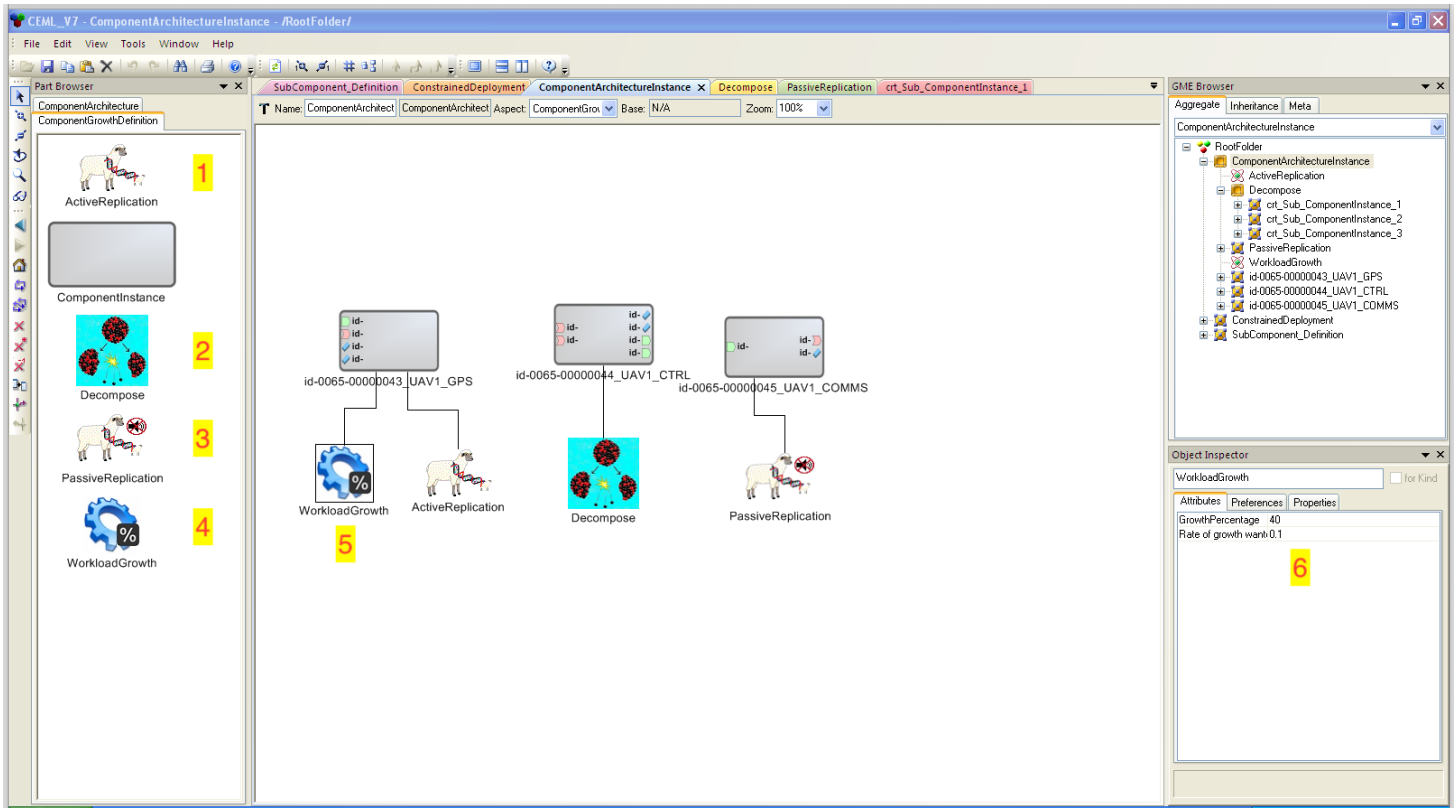

Figure G.2: Component growth modelling

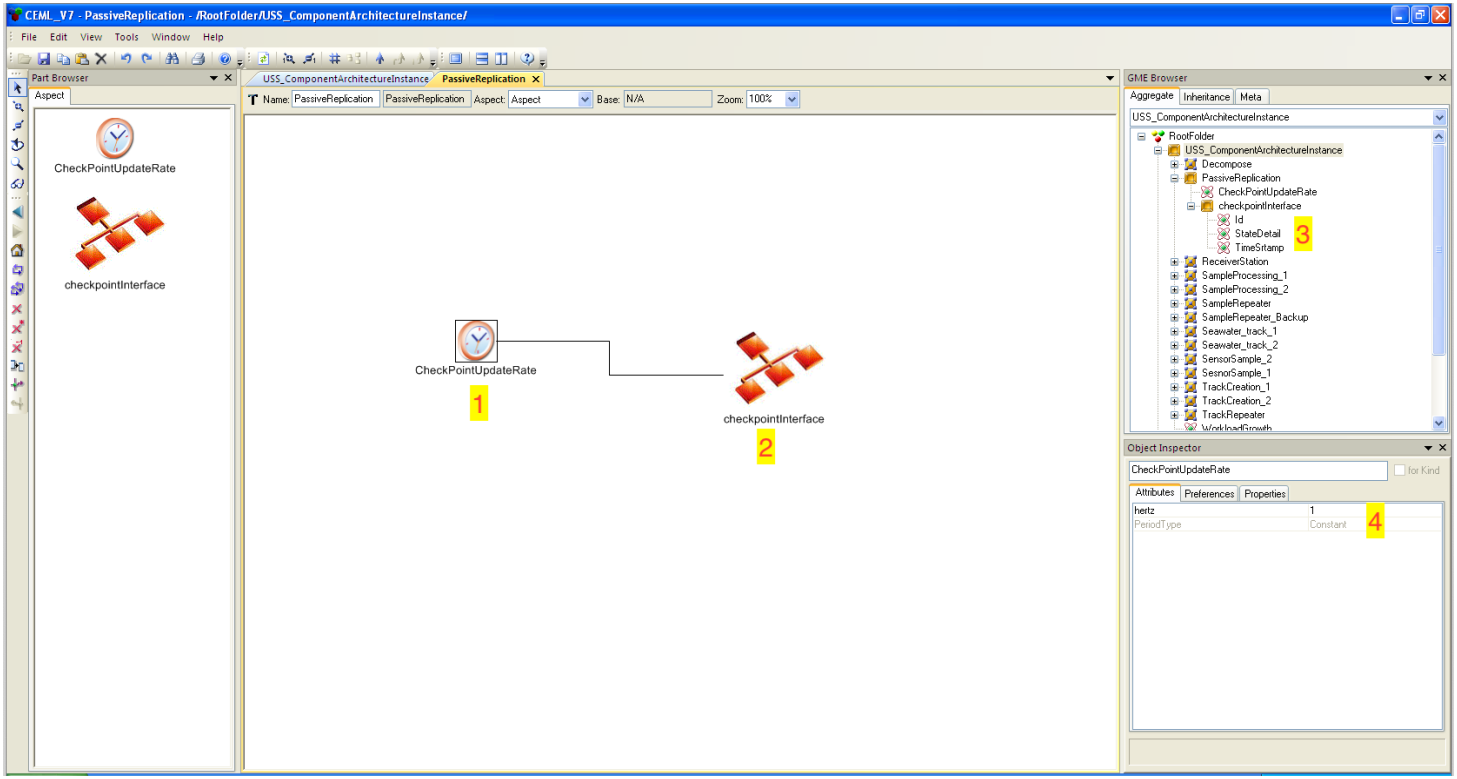

Figure G.3: Checkpoint modelling

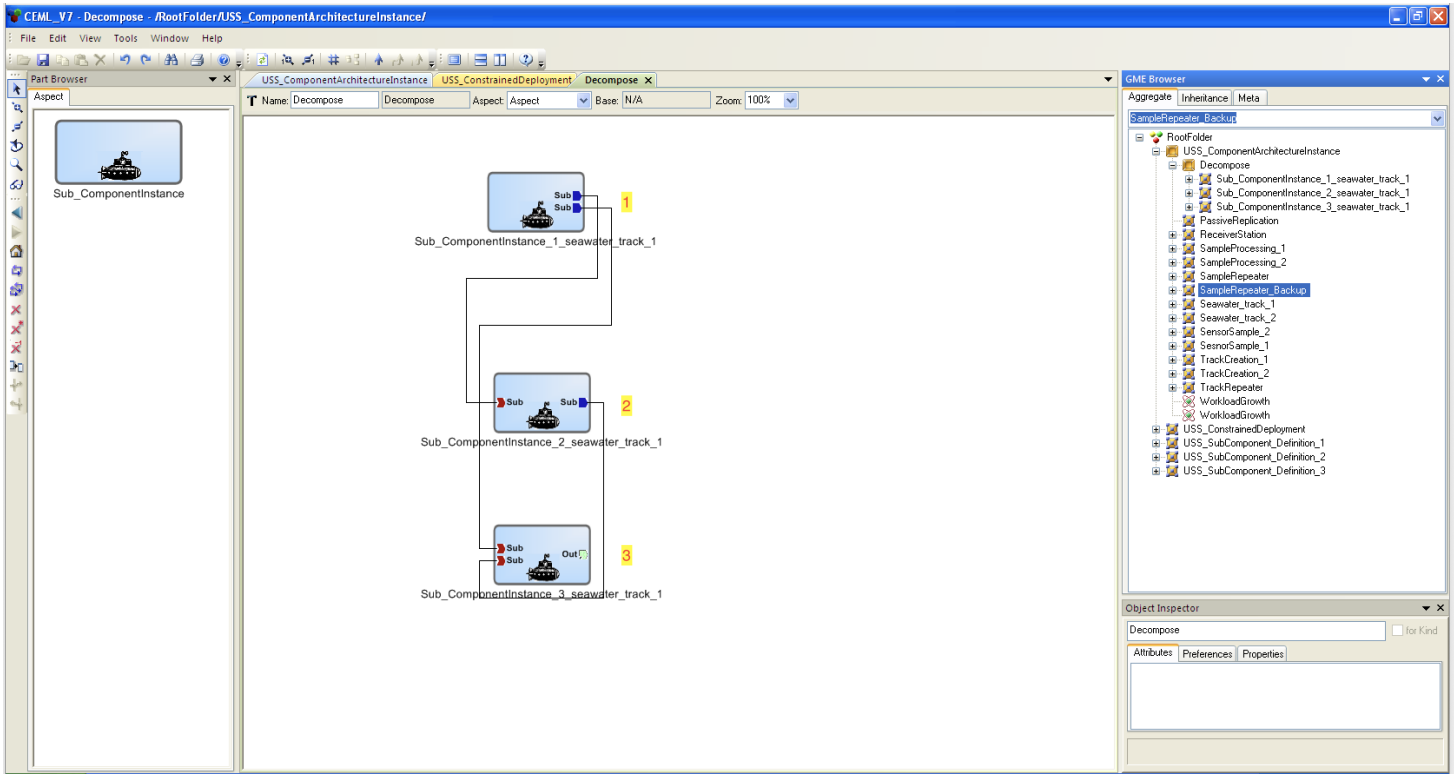

Figure G.4: Component decomposition modelling

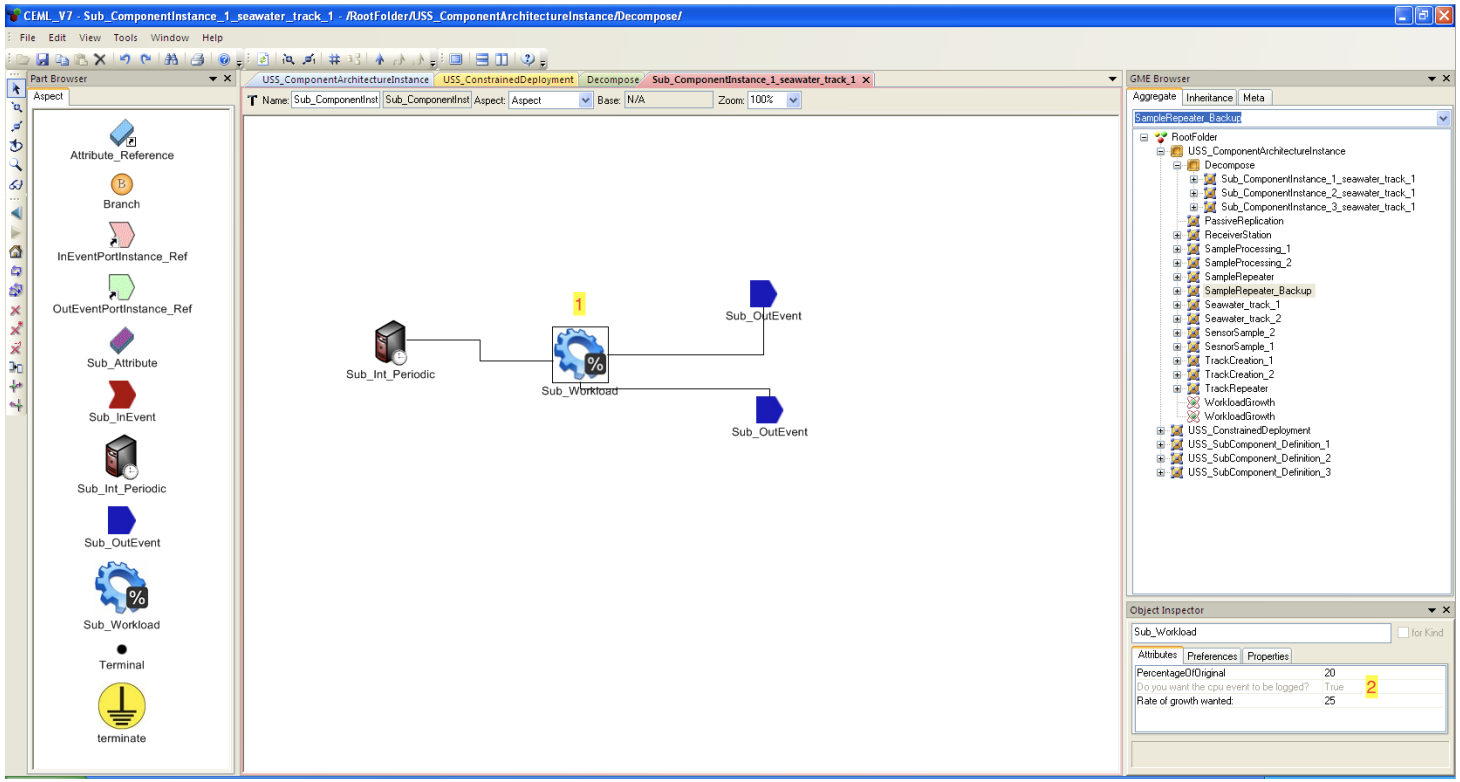

Figure G.5: Sub_ComponentInstance_1_seawater_track_1 behaviour modelling

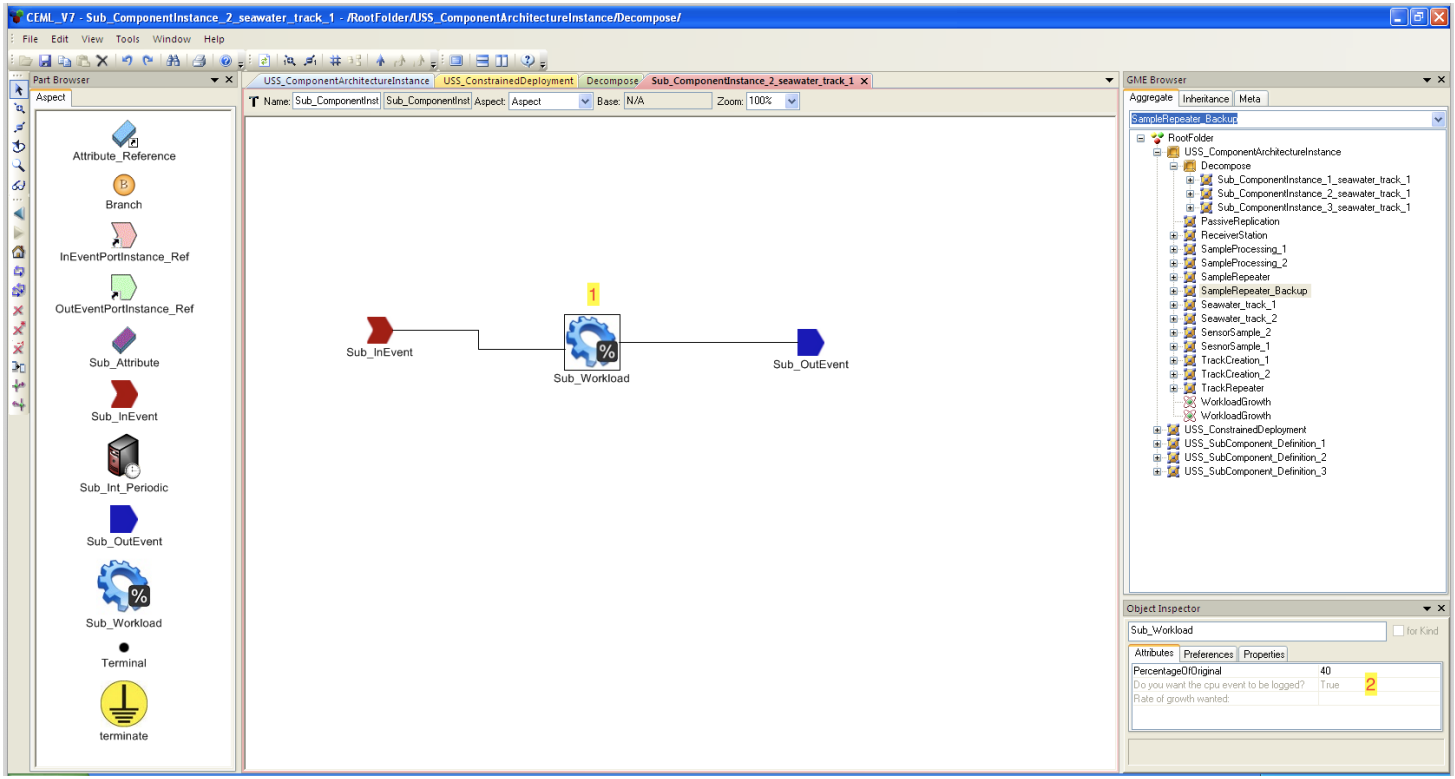

Figure G.6: Sub_ComponentInstance_2_seawater_track_1 behaviour modelling

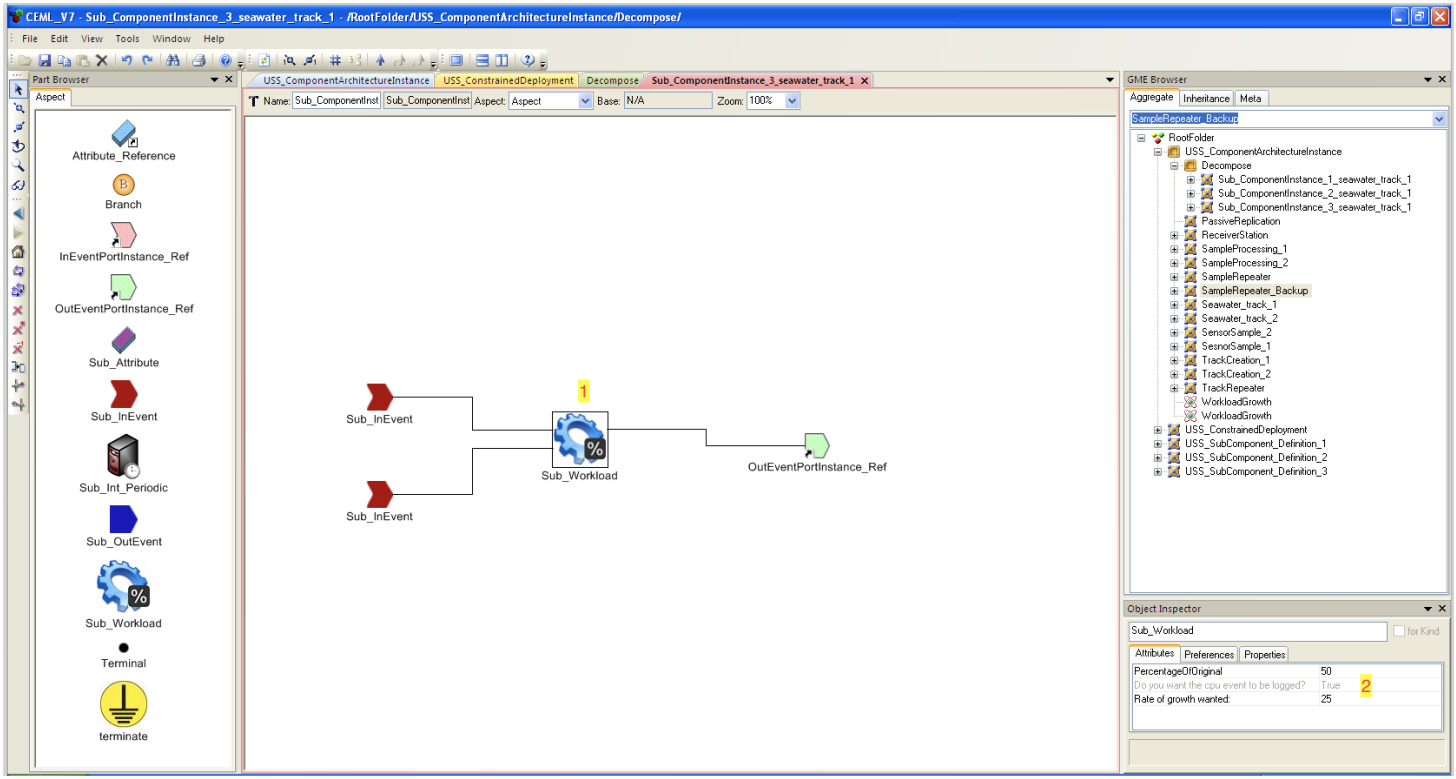

Figure G.7: Sub_ComponentInstance_3_seawater_track_1 behaviour modelling

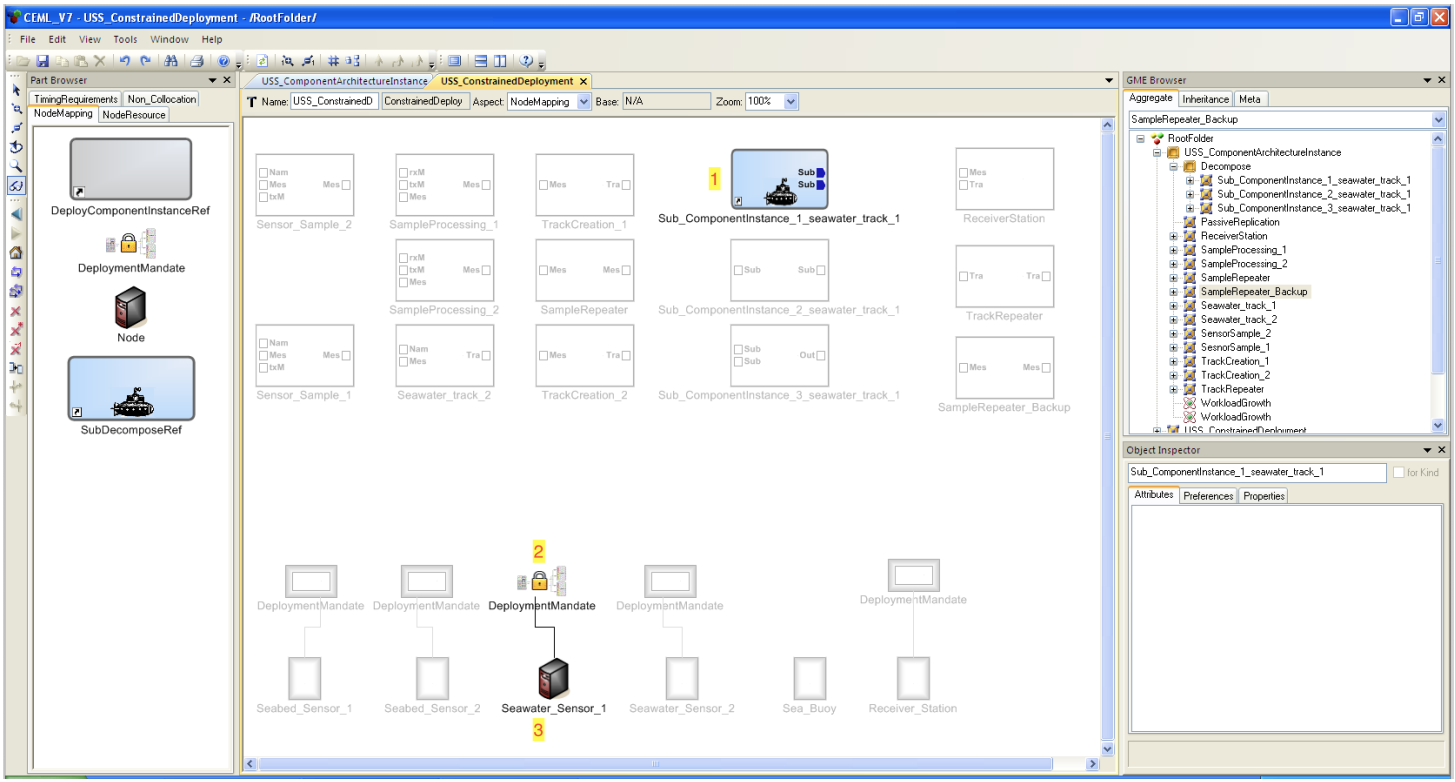

Figure G.8: New sub-component mandate deployment modelling

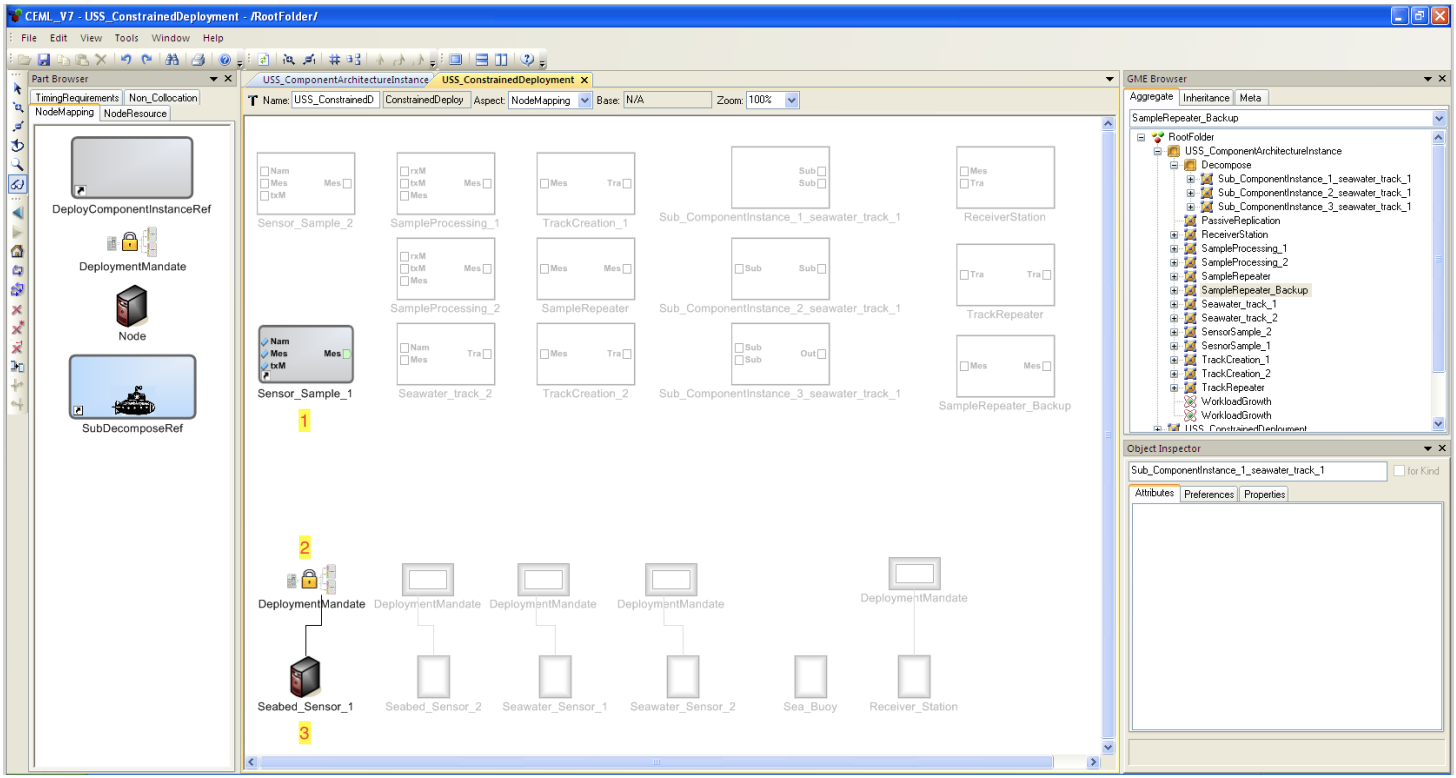

Figure G.9: Inherited component mandate deployment modelling

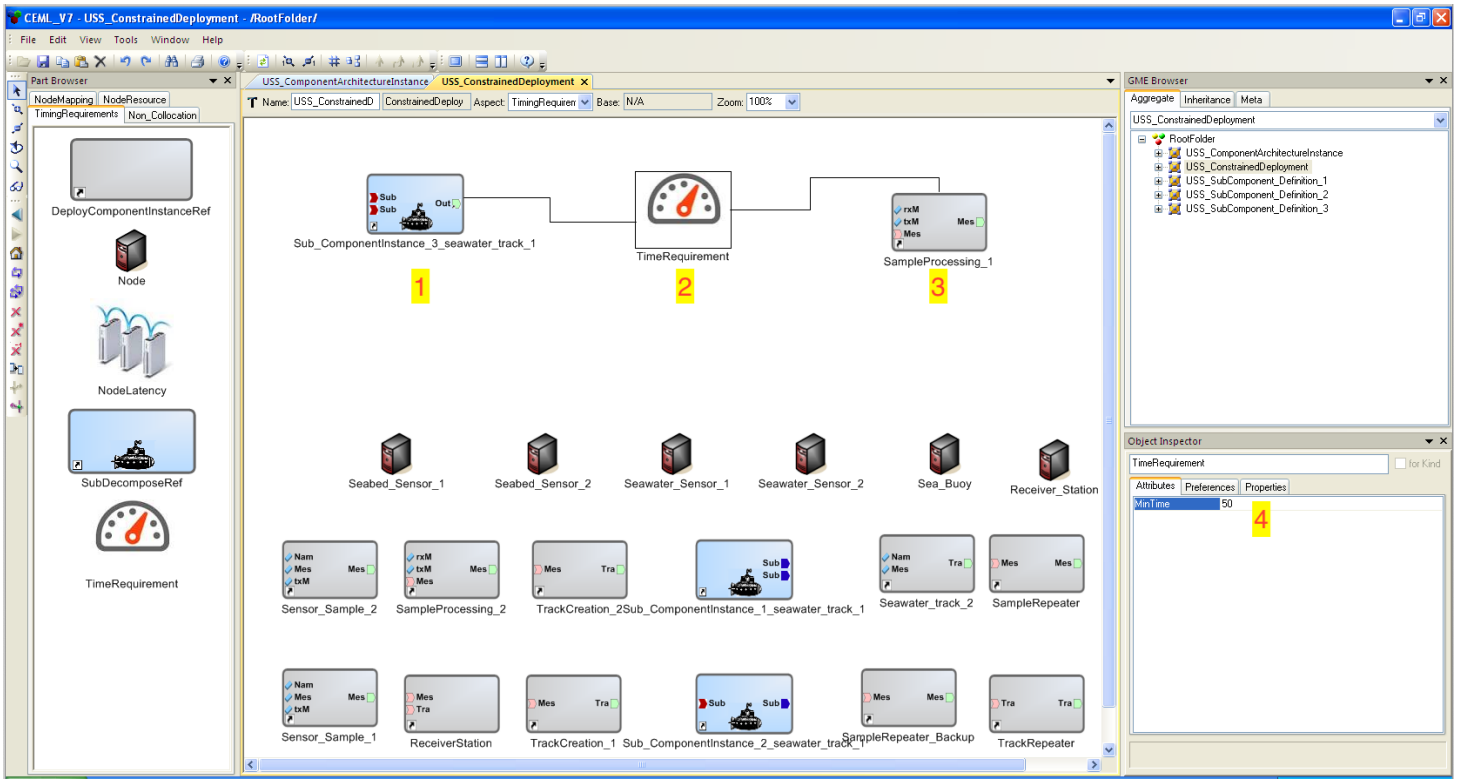

Figure G.10: Temporal performance constraint modelling

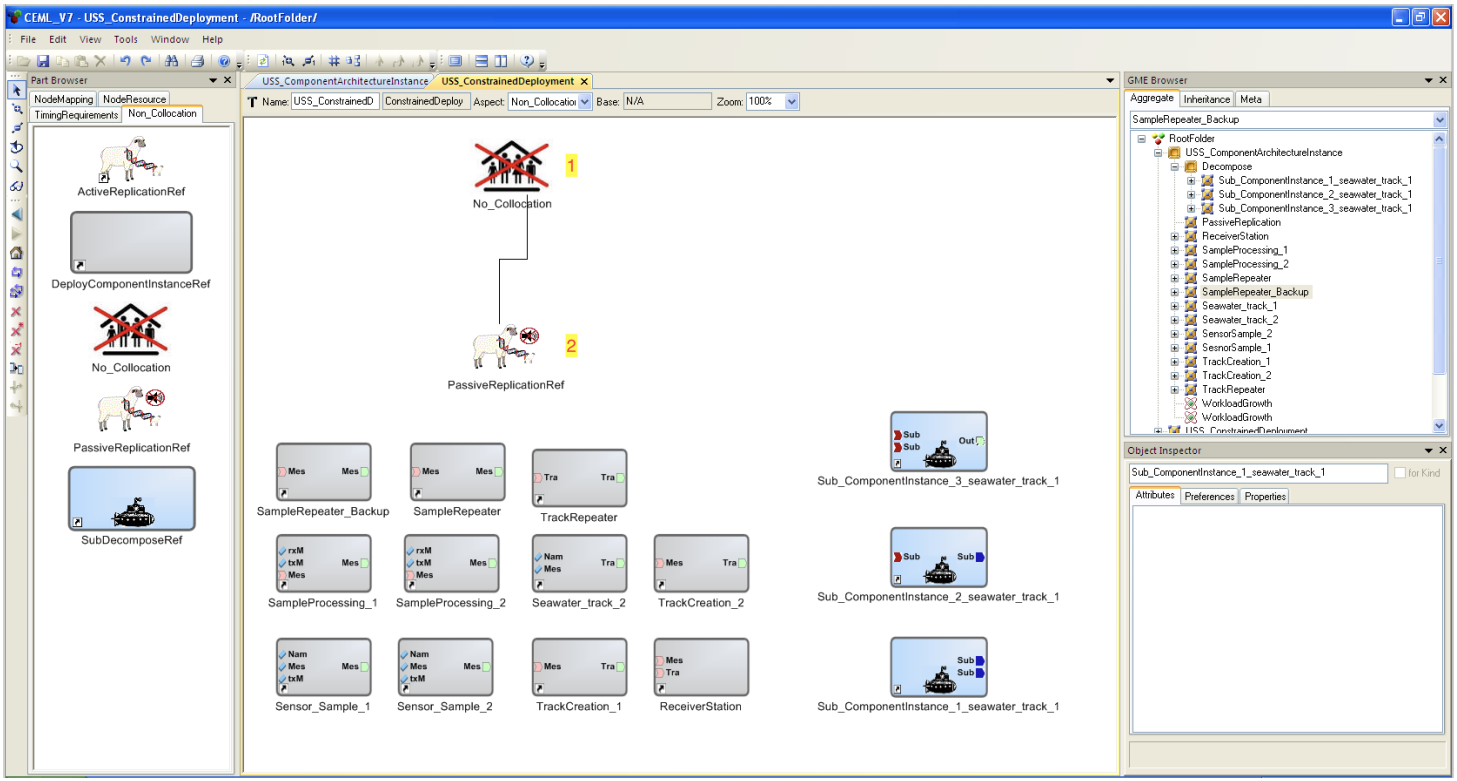

Figure G.11: No-collocation modelling

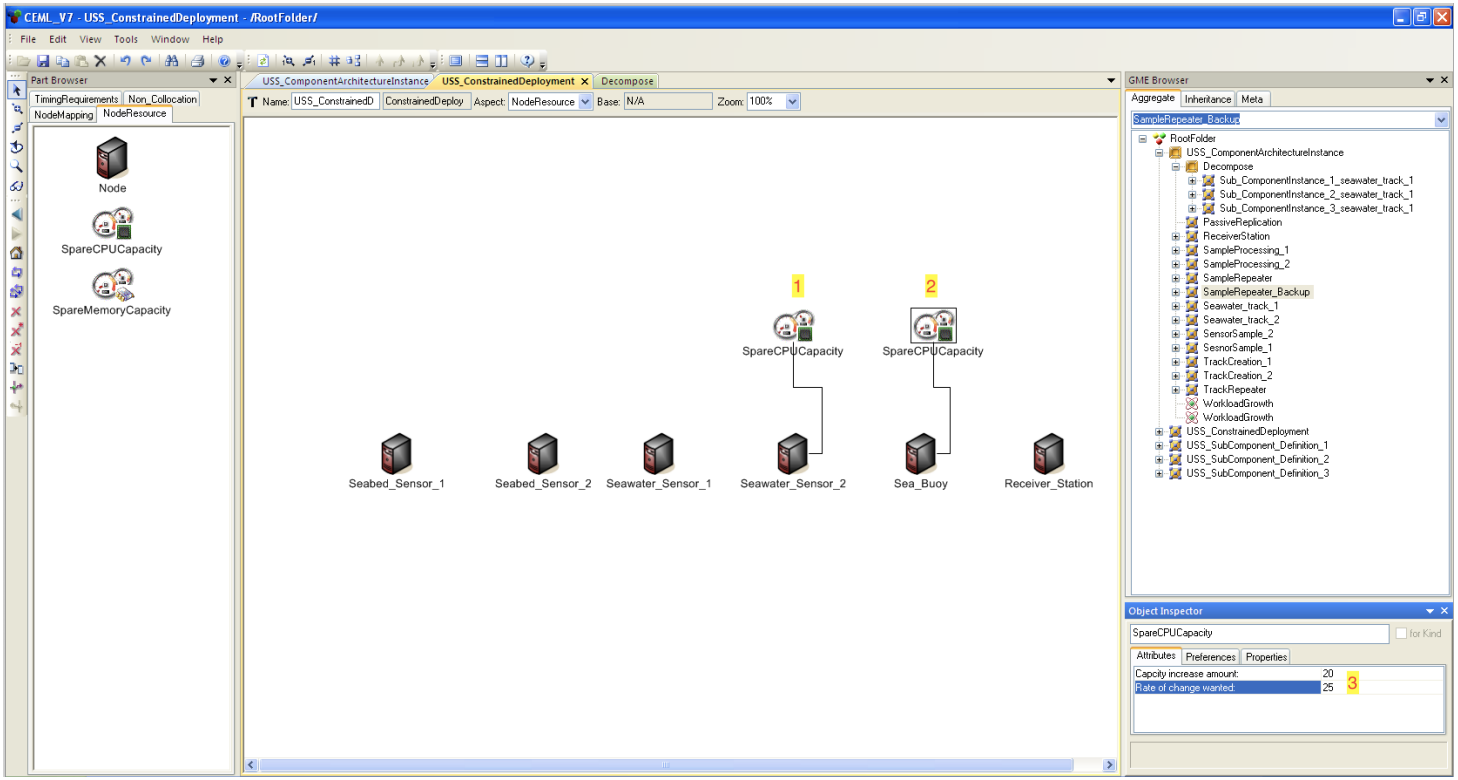

Figure G.12: CPU spare resource capacity modelling