Fortschritt-Berichte VDI

VDI

Reihe 8

Mess-,
Steuerungs- und
Regelungstechnik

Nr. 1267

M.Sc. Haitham Elfahaam,
Aachen

A Runtime Adaptation Concept to reinforce Versatility in Industrial Automation

Lehrstuhl für
Prozessleittechnik
der RWTH Aachen

A Runtime Adaptation Concept to reinforce Versatility in Industrial Automation

Der Fakultät für Georessourcen und Materialtechnik der
Rheinisch-Westfälischen Technischen Hochschule Aachen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

vorgelegte Dissertation

von

Haitham Elfahaam, M. Sc.

aus Giza, Ägypten

Berichter: Univ.-Prof. Dr.-Ing. Ulrich Epple
Univ.-Prof. Dr.-Ing. Birgit Vogel-Heuser

Fortschritt-Berichte VDI

Reihe 8

Mess-, Steuerungs- und Regelungstechnik

M.Sc. Haitham Elfahaam, Aachen

Nr. 1267

A Runtime Adaptation Concept to reinforce Versatility in Industrial Automation

Lehrstuhl für Prozessleittechnik der RWTH Aachen

Elfahaam, Haitham
A Runtime Adaptation Concept to reinforce Versatility in Industrial Automation
Fortschr.-Ber. VDI Reihe 08 Nr. 1267. Düsseldorf: VDI Verlag 2019.
130 Seiten, 63 Bilder, 10 Tabellen.
ISBN 978-3-18-526708-6 ISSN 0178-9546,
€ 52,00/VDI-Mitgliederpreis € 46,80.

Für die Dokumentation: Prozessleittechnik – Laufzeitadaption – Redeployment – Lastverteilung, Agentensysteme – Dezentrale Systeme – Industrie 4.0 – Wandelbarkeit – Optimierung – Stabilität

Unter dem Stichwort Wandelbarkeit wird die Fähigkeit verstanden, Industrieanlagen in die Lage zu versetzen, auf ungeplante Änderungen zu reagieren. Daher muss das Automatisierungssystem bereit sein auf Änderungen auf allen Ebenen in der Automatisierungspyramide reagieren zu können. In dieser Dissertation wird ein Konzept zur Laufzeitadaption vorgestellt. Das Konzept adressiert die Prozessleitebene und stellt ein Adaptionssystem vor, das die Softwarekomponenten im Netzwerk nach den verschiedenen Optimierungskriterien stabil verteilt und damit das Automatisierungssystem zur Wandelbarkeit befähigt.

Bibliographische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet unter www.dnb.de abrufbar.

Bibliographic information published by the Deutsche Bibliothek
(German National Library)
The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie (German National Bibliography); detailed bibliographic data is available via Internet at www.dnb.de.

D82 (Diss. RWTH Aachen University, 2019)
Tag der mündlichen Prüfung: 03. Juni 2019

ISSN 0178-9546
ISBN 978-3-18-526708-6

Preface

This dissertation was written during my employment as an academic researcher at the Chair of Process Control Engineering of RWTH Aachen University under the supervision of Prof. Dr.-Ing. Ulrich Epple and Prof. Dr.-Ing. Birgit Vogel-Heuser. The research was supported by the German Federal Ministry of Education and Research in the framework of the project BaSys4.0 (Förderkennzeichen 01—S16022).

I thank Prof. Epple for his mentoring, guidance and above all his trust that offered me freedom to think innovatively, encounter great experiences and gain exposure. His great insight and deep understanding of process automation and interdisciplinary research offered me a great chance to learn and grow as a researcher, for that I am and will be forever grateful. I also thank Prof. Vogel-Heuser for co-supervising my dissertation, for her keen remarks and careful revision.

I thank the former and current team members of the Chair of Process Control Engineering namely (in alphabetical order) Mahyar Azarmipour, Torben Deppe, Dr. Lars Evertz, Julian Grothoff, Holger Jeromin, Dr. David Kampert, Lars Nothdurft, Florian Palm, Christian von Trotha, Constantin Wagner for the fruitful discussions and cooperation. I also thank Margarete Milescu, Martina Uecker for their organization and efforts that helped me present this work.

I would like to express my deepest appreciation for the great work contributed by my student assistants Mariia Anapolska, Zolboo Erdenebayar and Michael Thies. Furthermore, I would like to express my sincere appreciation for the collaboration and fruitful discussions with project partners, Dr. Sten Grüner and Tarik Terzimehic.

I am immensely grateful to my parents, my brother, my grandmother and my family for their unconditional love, support and for being there through thick and thin.

Finally, I dedicate my dissertation to my late grandfather who helped and wished to see this work come to light. May your memory be eternal.

Haitham Elfaham
Aachen, March 2019

Contents

Abstract

In the process control engineering domain, various initiatives around the world (e.g. "Industry 4.0" in Germany) play a crucial role in directing the research and development. In the new generation of industrial automation, a new architecture is introduced where the communication hierarchy of the automation pyramid is dissolved in order to increase the flexibility of the production systems. One of the objectives of this architecture is to achieve "Adaptability" or in other words to enable industrial plants to react to unplanned changes. Furthermore, design principles like decentral decision making and interconnectedness are widely promoted. In order to achieve the aforementioned goals, various new functionalities (e.g., Self-X functionalities and optimizations) and information (e.g., asset administration shell) are being introduced to the current production systems which did not exist in the conventional ones thus causing a dynamic overhead to the available resources (computation, communication, dynamic memory, etc.).

In the conventional systems, during the engineering phase, control logic and functionalities are designed and then deployed to the computation nodes in the automation network. In some cases, an optimized distribution profile for the loads are computed prior to the initial deployment and accordingly the load is distributed amongst the network endpoints. However, the dynamic aspect of the load variations introduced in the newly introduced automation paradigm is not taken into consideration.

System adaptation to the varying loads is required to readjust the loads and balance the resources consumption in the network. In industrial automation, safety aspects play a crucial role. Hence, a prerequisite for this framework is to not compromise the stability of the production system.

The objective of this dissertation is to establish a framework for a seamless integration of a deployment platform that can, through redeployment and adjustment of software components, balance the resources consumption overhead amongst the automation network participants, establish redundancy of the different components, improve the communication quality of service and adapt the system according to the rapid and dynamic changes imposed.

Thorough analysis and investigations for stability and the production system dynamics are conducted. The goal of these investigations is to ensure that the introduced framework does not affect the performance in any undesired manner, e.g., causing the loads to oscillate in the network or affecting the system performance with a non converging redeployment processes of the software components. Hence, additional to these investigations, a multi optimization criteria load balancing model is constructed to investigate the behavior or multidimensional optimizations. Moreover, performance enhancements analysis is conducted through investigating the automation networks and constructing regression models to compute the optimal parameters for load redeployment.

Furthermore, a prototype implementation to reinforce the presented concepts and validate the conducted investigations is realized. The prototype considers an aluminum cold rolling mill use-case and utilizes two different approaches namely decentral algorithms and

agent systems approaches to perform the load balancing from two different perspective namely resources and component perspectives respectively. In the former approach, the algorithm uses mathematical formulas (e.g., total square error) to compute the optimum load balancing profile from a decentral perspective and cooperates with other network participants to achieve the optimum load distribution profile on a global scale. On the other hand, in the latter approach, the components are considered as independent agents that wander the network. The information incubated within an agent is used (e.g. optimal routed path according to a given recipe) to anticipate the load distribution in the network and thus adjust the placement of the components (agents) accordingly.

The presented prototype implementation uses the runtime environment ACPLT/RTE and acts as extension library to provide the load balancing functionalities. The implementation uses the demonstrator from the SMS-group that simulates a cold rolling mill plant.

Kurzfassung

Die Prozessleittechnik als ein Teil der Automatisierungstechnik erfährt durch Initiativen wie „Industrie 4.0“. In der Prozessleittechnik sind verschiedene Initiativen auf der ganzen Welt (z. B. „Industrie 4.0“ in Deutschland) derzeit in Forschung und Entwicklung richtungsweisend. Die Auflösung der Kommunikationshierarchie in der Automatisierungspyramide wird als zielführend erachtet, um den zunehmenden Anforderungen an Flexibilität in Produktionssystemen gerecht zu werden. Unter dem Stichwort „Wandelbarkeit“ wird die Fähigkeit verstanden, Industrieanlagen in die Lage zu versetzen, auf ungeplante Änderungen zu reagieren. Ansätze dazu sind Gestaltungsprinzipien wie dezentrale Entscheidungsfindung und vollständige Vernetzung. Um die zuvor genannten Ziele zu erreichen, werden verschiedene neue Funktionalitäten (z. B. Self-X-Funktionalitäten und -Optimierungen) und Informationen (z. B. die Verwaltungsschale) in die derzeitigen Produktionssysteme eingeführt, wodurch eine Dynamik und ein Overhead zu den verfügbaren Ressourcen (Berechnung, Kommunikation, dynamischer Speicher usw.) erzeugt wird.
In den konventionellen Systemen werden während der Engineering-Phase Steuerlogik und Funktionalitäten entworfen und dann an die Rechenknoten im Automatisierungsnetzwerk verteilt. In einigen Fällen wird ein optimiertes Verteilungsprofil für die Lasten vor der ersten Bereitstellung berechnet, und dementsprechend wird die Last auf die Netzwerkendpunkte verteilt. Der dynamische Aspekt der Lastschwankungen des neu eingeführten Automatisierungsparadigmas wird jedoch nicht berücksichtigt. Eine Systemanpassung an die unterschiedlichen Lasten ist erforderlich, um die Lasten neu verteilen und den Ressourcenverbrauch im Netzwerk auszugleichen. In der industriellen Automatisierung spielen Sicherheitsaspekte eine entscheidende Rolle. Eine Voraussetzung für diesen Rahmen ist daher, die Stabilität des Produktionssystems nicht zu beeinträchtigen.
Das Ziel dieser Dissertation ist die Schaffung eines Rahmens für die nahtlose Integration einer Implementierungsplattform, die durch die erneute Bereitstellung und Anpassung von Softwarekomponenten den Ressourcenverbrauch zwischen Teilnehmern eines Automatisierungsnetzwerks ausgleicht, die Redundanz der verschiedenen Komponenten herstellt und die Kommunikation verbessert in Bezug auf Servicequalität und sowie die Anpassung des Systems an die schnellen und kontinuierlichen Änderungen.
Gründliche Analysen und Untersuchungen zur Stabilität und zur Dynamik des Produktionssystems werden durchgeführt. Das Ziel dieser Untersuchungen besteht darin, sicherzustellen, dass das eingeführte Framework die Leistung nicht auf unerwünschte Weise beeinflusst, z. B. indem die Lasten im Netzwerk oszillieren oder die Systemleistung durch nicht konvergierende Umverteilungsprozesse der Softwarekomponenten beeinflusst wird. Zusätzlich zu diesen Untersuchungen wird daher ein Lastausgleichsmodell für mehrere Optimierungskriterien erstellt, um das Verhalten oder mehrdimensionale Optimierungen zu untersuchen. Darüber hinaus wird die Analyse der Leistungsverbesserungen durchgeführt, indem die Automatisierungsnetzwerke untersucht und Regressionsmodelle erstellt werden, um die optimalen Parameter für die Lastumschichtung zu berechnen. Darüber hinaus wird eine Prototyp-Implementierung realisiert, um die vorgestellten Konzepte zu verstärken und

die durchgeführten Untersuchungen zu validieren. Der Prototyp betrachtet den Anwendungsfall eines Aluminium-Kaltwalzwerks und verwendet zwei unterschiedliche Ansätze, nämlich dezentrale Algorithmen und Ansätze von Agentensystemen, um den Lastausgleich unter zwei verschiedenen Gesichtspunkten durchzuführen, nämlich Ressourcen- und Komponentenperspektiven.
Die vorgestellte Prototypimplementierung verwendet die Laufzeitumgebung ACPLT/RTE und fungiert als Erweiterungsbibliothek, um die Lastenausgleichsfunktionen bereitzustellen. Die Implementierung verwendet den Demonstrator der SMS-Gruppe, der eine Aluminium-Kaltwalzanlage simuliert.

1 Introduction

The German initiative “Industry 4.0” (I40) has gained a lot of attention in the automation community. Topics like cyber physical systems, cloud computing and Internet of Things (IoT) play a crucial role in the initiative [36]. I40 aims to introduce a fourth industrial revolution that establishes a paradigm shift in the classical automation infrastructure by promoting computerization of manufacturing. A challenge that is faced in I40 is the “Wandelbarkeit” or adaptability of the plant to unplanned changes in the environment and conditions [3] [82]. In Industry 3.0 or the classic infrastructure, there exists a communication hierarchy between the different layers of the automation shown in Fig. 1.1. This can hinder the interconnectedness between all layers and consequently the concept of adaptability. Adaptability describes the ability to accommodate unplanned disruptions or changes [1]. One of the objectives of I40 is to dissolve the communication hierarchy and offer a more interoperable environment through higher interconnectedness between the industrial systems on the different layers.

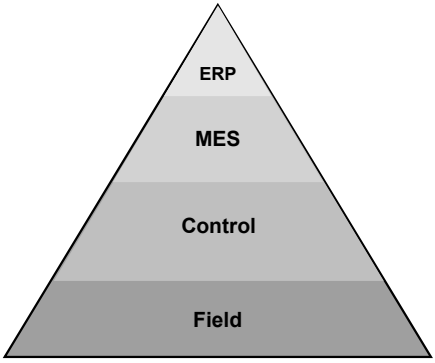


Figure 1.1: Automation Pyramid

1.1 Motivation

In the I40 automation-paradigm, the environment is constantly changing in a highly dynamic pace pertaining to all layers in the automation pyramid. Contrary to the classic paradigm for instance, where initially the resources and the tasks are determinately distributed in the planning stage and the strategy is held for long times until claimed otherwise [93]. Although this might be adequate for the classic paradigm, nowadays resources, capabilities and even the tasks dynamically change requiring a continuous adaptation of the

set plan so that it can accommodate to the newly introduced modifications [79]. In order to accomplish this, a strategy is required to analyze an automation network and attain the required adaptation objectives. In this contribution, a concept is introduced with a prototype implementation to demonstrate how runtime adaption using redeployment can enhance and optimize the resources utilization in the automation domain.

1.1.1 Applications and Scenarios

Scenarios and applications that can be attained by the redeployment process:

- **Availability:** Having such dynamic changes dictates a continuous monitoring of the load change over the network and consequently a redistribution of the load such that there is enough resources reserve at each node. Resources in that sense can be seen as the computation, communication or storage power whereas load is seen to be the software components being redistributed. A unified load distribution and consequently a unified resources reserve distribution can increase the readiness of each node to react in critical situations.

- **Redundancy:** Maintaining a certain degree of redundancy in a network by producing backup copies of a component and distributing copies in the network.

- **Node Outage:** Ensuring an uninterrupted execution in the case of a node outage. Whether the node outage is planned due to maintenance or an unexpected due to failure, the components running on that node as well as their states can be redeployed on another node provided that the states are synchronized and the components either are stored in the cloud or a redundant version exists in other nodes.

- **Communication Quality of Service (QoS):** The quality of communication can be enhanced through reducing the communication overhead. The overhead reduction can be established either by reducing the distance between two communication endpoints thus guaranteeing a better QoS, or by (temporarily) reducing the frequency of requests produced by other components in the communication path reducing the bandwidth depletion.

- **Process Control Component Upgrade/Replacement:** Upgrading or replacing an already available PCC with another from the cloud. This can take place due to different reasons, e.g., purchasing an upgraded functionality or modification of the planned role of a device, updating the control logic, etc.

- **Functionality Enhancement:** Some components are equipped with additional tools that caters luxurious functionalities, e.g., achieving higher precision than required. In critical conditions during redeployment, these functionalities are usually disabled as an adjustment procedure to increase the resources reserves at a node to provide slack for other components to be deployed. However, after handling through the critical conditions, the adjustment procedures are not rectified to normal conditions again. Redeployment can increase the resources reserves at a node by redeploying the components available at the node to another node. Providing the maximum resources to be depleted by the component enables the component to reach its highest precision as well as to activate luxury functionalities.

- **Boundary Conditions:** A boundary condition here can indicate hardware dependencies, e.g., the availability of certain communication infrastructure, real time requirements, e.g., after upgrading a process control component, the current communication channel bandwidth cannot support the required QoS, etc.

- **Parallelization:** Redeploying branches in PCCs that can be parallelized (e.g., parallel branching in sequential state charts) on other computation nodes to enhance processing performance.

1.1.2 Problem Definition

The problem can be primarily seen in the lack of autonomy in the present automation systems. This lack of autonomy is due to the absence of the interrelationships between the available tools and established information. For example, nowadays, the systems are engineered according to fixed recipes and reacting to changes and the dynamic modification in the shop floor is limited. During the engineering phase, engineering costs and months are spent designing and developing control logic for devices that are later deployed on fixed computation endpoints. However, during the production process, the conditions are dynamically changing which results in continuous change of the loads (i.e., computation, communication or dynamic memory loads) in the network. Furthermore, computing an optimal plausible solution for the allocation problem of the given software components is a complicated task and is proven to be an NP-hard problem [61]. Establishing an autonomous system that can redistribute the load in the network is a crucial prerequisite for adaptability in I40. The adaption systems can precisely evaluate the dynamic situations the network is experiencing and adapts it by adjusting the load in the network accordingly. Redeployment can offer a solution that increases the resources reserve in the computation nodes and eventually the readiness of the system for critical situations.

1.2 Objective of this Work

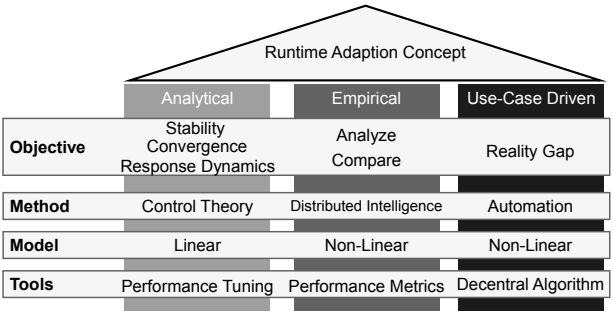


Figure 1.2: Concept Overview

The objective of this dissertation is to introduce a concept that establishes a runtime adaption system that optimizes the utilization of the available resources in dynamic conditions. The optimization is achieved through the rearrangement of the components in the network ensuring an optimal distribution in terms of the resources utilization. The introduced concept focuses on industrial automation scenarios and is demonstrated on a use-case in discrete manufacturing. Furthermore, decentral control and system autonomy are considered as design requirements. The former sustains the Industry 4.0 design principle decentral decisions. The latter ensures a minimal human intervention reducing personnel overheads. The concept considers three fundamental perspectives as shown in Fig. 1.2:

- Analytical Perspective: During load balancing, coordinating different optimization objectives (e.g., load distribution, redundancy insurance, etc.) can possibly introduce undesired behaviors (e.g., load oscillations) or even compromise the stability of the system. In this perspective, the following objectives are investigated: maintaining system stability and state convergence, controlling system response dynamics and enhancing its performance. A formal model to describe the system introduces an analytical perspective of the load balancing problem hence a control theory methodology is utilized. A linear model is introduced in this perspective which provides a tuning tool for performance enhancement as a result.

- Empirical perspective: Various decentral algorithms in the informatics and system engineering domain address the load balancing problem. However, due to nonlinearity in the system, it is difficult to model the load balancing problem on that level. Thus, in order to analyze the existing algorithms and also to evaluate their performance under different initial and boundary conditions (e.g., initial load distributions, network topologies, network sizes, etc.), a performance metrics list is defined. The list produces a comparative scheme for the different algorithms which helps understand the advantages of the algorithms in the different conditions.

- Use-case driven perspective: Through the acquired knowledge, an algorithm can be developed and implemented in the industrial automation domain. However, due to the reality gap, it is essential to examine a use-case from an industrial automation perspective which considers the nature of the software applications (e.g., control hierarchy), communication protocols and real time requirements. A demonstrator that simulates an aluminum cold rolling mill plant is considered to assess the developed concept.

1.3 Structure of the Dissertation

The rest of the dissertation is structured as follows:

- In Chapter 2, the state of the art is presented. In this chapter, relevant work, important concepts and previously published contributions that are used as preliminary work or lie in the scope of this dissertation are discussed. Additionally, an overview is given for the technologies utilized in the implementation.

- Chapter 3 explains the presented adaption concept thoroughly. An overview of the concept of adaption and redeployment in automation and how it can be realized are presented. In this chapter, only a collective perspective is discussed. Nevertheless, exclusive separate chapters are assigned to discuss the analytical areas in more details.
- In Chapter 4, the modeling fundamentals are discussed. An analytical abstract model is discussed. This model is adjusted to fit the different approaches discussed in the following chapters.
- In Chapter 5 and 6, analytical and empirical approaches of the redeployment process are presented respectively. In the former approach, the dynamics of the system response and the performance enhancements are discussed. While in the latter, a performance analysis from a software perspective is presented and a benchmark to test the different available algorithms is introduced.
- Chapter 7 discusses a realistic perspective of the problem by introducing an aluminum cold rolling mill use-case.
- Chapter 8 presents the two approaches for the realistic perspective namely the decentralized algorithm and the agents systems approach respectively.
- Chapter 9 and 10 demonstrate the prototype implementations for both approaches, decentralized algorithm and agents systems, respectively. Furthermore, testing of the different functionalities provided is presented as well as the results delivered by the implementations.
- Chapter 11 concludes the dissertation and gives an outlook for the presented work.

2 State of the Art

The presented work combines various concepts from different disciplines, hence the wide diversity of topics presented in this chapter. Each section provides either an overview for previous work that was done within the focus of this contribution or an insight into a utilized technology or tool.

2.1 Dynamic RunTime Environments (RTE)

In process control applications, the functional realization typically takes place on the basis of the function block concept of IEC 61131 [87]. As shown in Fig. 2.1, this concept has a development phase (where programming of libraries is performed), as well as an operating phase in the life cycle of the automation software (where components and systems are engineered). In the development phase, new function block types are created [97]. This can be conducted using a variety of languages. In many cases this is done by the manufacturer of the automation system. The result is ready-made, checked and verified libraries of function module types. In the engineering phase, instances of these types are created and interconnected through communication relationships in such a way that the desired application functionality arises. Especially in the field of process control technology, real-time functional block operating systems have been developed from the outset, which allow engineering and re-engineering of the block structure at runtime. These operating systems are also referred to as dynamic runtime environments where instances and connections can be recreated, reconfigured, or deleted at runtime [94]. This is done on the basis of loaded type libraries. Rebuilding or modifying type libraries is generally not possible at runtime. There are a few exceptions: some systems allow the modification of special types written for example in the Structured Text (ST) programming language, while others allow for dynamic reloading of libraries. The ability to reconfigure the instance system at runtime also opens up the possibility to dynamically move functionality between the system components and is therefore a prerequisite for dynamic redeployment. However, there are a number of boundary conditions to consider like the compatibility of the underlying platform and other conditions that are covered later in Sec. 3.4.2. Underlying platforms must provide a basis layer to ensure compatibility, thus allowing an undisrupted redeployment operation. Hypervisors (type 1 or 2) and Containerization technology offer a good preliminary work to design and construct the underlying layer for redeployment (cf. Fig. 2.2).

2.2 Virtual Machines

Virtualization is the abstraction of functional realization from the concrete assignment to software and hardware objects. The best known form of virtualization is the Virtual Machine (VM). A VM encapsulates an operating system within a base operating system.

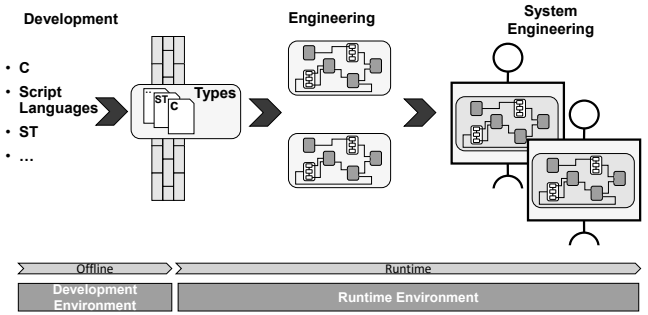


Figure 2.1: Phases of Development, Engineering and Operation of Applications

A VM system platform allows the handling of complete operating systems with the applications implemented on them as encapsulated components. Such a component can be exchanged between different nodes of a VM system. The prerequisite is the compatibility of the underlying basic operating systems. While the components are securely encapsulated in content by the VM system, from the point of view of real-time requirements crucial weaknesses, it is the sole responsibility of the underlying base system to organize the allocation of resources (computing time, storage space, communication channels) and to guarantee the applications. So the usual operating systems (RT Linux, etc.), for example, as basic systems do not guarantee the execution of the different VM in real time. Hypervisor solutions can provide a workaround, as basis operating systems that are able to provide such guarantees to their container-packed VMs [7].

2.3 Container Technology

Containerization offers a medium for deployment which is used to inspire the concept presented in the dissertation. The term container has been used in different concepts. In order to avoid confusion, Fig. 2.2 shows the three different types of containers. In the first form, a hypervisor container is shown. A hypervisor container (sometimes referred to as a partition as well) is used when a hypervisor of type 1 is utilized. The second form of containers also known as a virtual machine is when hypervisor of type 2 is utilized. The difference is that the former type is in direct contact with the hardware and distributes the resources amongst the Operating Systems (OS) running on it, while the latter is running directly on the operating system layer and emulates a different system architecture. The third type is a deployment container that does not pack any operating systems. The prerequisite for this container is the installation of the container administration platform on all nodes participating in the deployment. Container virtualization has been used more and more for e-commerce in recent years (PayPal, VISA [25]) because it offers the possibility of application-level rather than virtual machine-based operating system-level support to virtualize. A distinct advantage is the fact that resource consumption can be significantly reduced in terms of memory and computational load, as a single kernel is shared by multiple containers. Within a container, only the necessary dependencies such as libraries, tools,

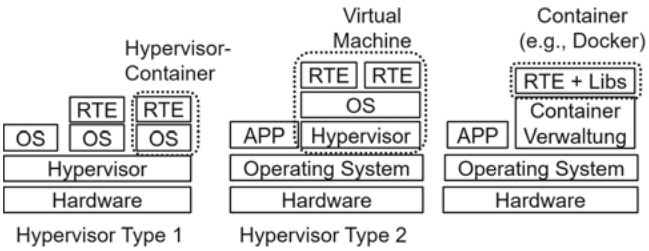


Figure 2.2: Illustration of the different containers

and configurations are packed (see Fig. 2.2). Similar to a VM, containers isolate the virtual environment from other containers. This makes it possible to run multiple applications in separate containers on a host system and still meet security requirements [25]. Docker is a popular and widespread implementation for container virtualization that simultaneously provides an ecosystem with the necessary management functionality for the deployment and operation of distributed containers. For this reason, containerization with Docker is exclusively considered. Furthermore, Docker offers various tools, e.g., load distribution and redundancy mechanism, not only the containerization technology. Essential elements of Docker are listed below and explained in more detail:

2.3.1 Docker-Daemon

The deployment requires the presence of a Docker platform (daemon). The daemon allows the download and upload of containers from and to a repository (Docker cloud). In the optimization domain, Docker utilizes a consensus algorithm called "Raft" [68] in its swarm mode for electing master nodes that can administer redundancy and load distribution operations centrally.

2.3.2 Load Distribution

In the "Swarm" mode, Docker can structure the computing capacities (computer nodes) into so-called groups of swarms. The nodes can be set as a Master or as a Worker. The load is either uniformly or redundantly distributed to the nodes according to the specified setting [84].

2.3.3 Compatibility

Container technology enables deployment between different systems that have different infrastructures. To do this, a container must contain all libraries needed to capture all dependencies. In the case of different operating systems between the host and the guest, the container can contain a minimal image of the operating system and can thus function in principle as a VM.

2.4 Migration and States Synchronization

In case of performing a redeployment of a component that has internal states, a synchronization of the states is necessary to ensure an uninterrupted and a stable handover between the deployed and the original component. A drawback can be seen in some cases while establishing a communication channel with a certain Quality of Service (QoS) between the components. This requires the handling of network communication channels of the Docker container which can pose some limitations.

The migration is measured by two crucial metrics. Firstly, the downtime which describes the duration during migration where the service is in a completely halted state. Secondly, the migration time which corresponds to the time duration from triggering the migration process till the service full functionality restoration on the destination server. The down time is surely a more important metric as not meeting the industrial application response time requirement can jeopardize the process control stability [19]. In this section, the different exploited concept to perform a state synchronization are discussed. The terms source server and destination server are used to refer to the servers processing the component before and after migration respectively.

2.4.1 Service Migration in Automation

In automation, migrating a service is performed over two main stages: migrating the service itself and the memory it uses to synchronize the states. The service itself is considered static and is not altered during runtime. On the contrary, the memory of a service is dynamic and is constantly changing during an execution. Different mechanisms exist that perform service migrations. The mechanisms differ in terms of the sequence of execution, e.g., copying sequence of the service and the memory (which is performed first and which is second). The following mechanisms demonstrate the proposed mechanisms in the literature:

Pre-copy Migration

This scheme proposes a two phase migration, push phase and stop-and-copy phase. In the push phase, the corresponding memory, that the service uses, is copied. The copying process is performed on the fly, i.e, the memory pages can be altered. Hence, the copying process is performed iteratively until either the number of alteration reduces to or exceeds a certain threshold [86]. The stop-and-copy phase is when the downtime occurs. Here the source server shortly halts the service to ensure state synchronization consistency. During this time, the service becomes unavailable and is restored on the destination server upon successful synchronization.

Post-copy Migration

Similar to the Pre-copy migration, this scheme comprises two stages, stop-and-copy and pull Phases. However, the post-copy executes the migration in an opposite sequence. The freezing of the service occurs in the initial phase when only the state of the CPU is transferred to the destination server and not the service itself [48]. The destination server restores and resumes the service, albeit experiencing a degraded service due to glitches of

page faults during requests of memory pages that are not yet retrieved from the source server. At this point, the pull phase is initiated requesting the memory pages causing a jittery response as well as unpredictable response times.

Hybrid

In [57], endeavors to combine the complementing performances of the aforementioned Pre- and Post-copy migration schemes using the hybrid scheme. This scheme introduces a three phase migration namely push, stop-and-copy and pull phases. The phases operate in the same manner as described in the two previous sections.

Disruption-free Software Updates

Further approaches are presented in [83] and [95]. In [95], a controller of an automation system without disrupting the system's operation is updated. As a use case, the author uses a magnetic levitation example application to validate the approach. The controller is synchronized and a handover is performed.

2.4.2 Redundancy Migration

Redundancy migration surpasses the performance of the aforementioned methods in terms of downtime [42]. This migration scheme is evaluated using LinuX Containers (LXC). It comprises four phases namely Buffer and Routing initialization phase, Copy and Restore phase, Replay phase and Switch phase. In the first phase, when the migration is triggered, the communication is rerouted such that the client messages are no longer directly sent to the source server rather via the destination server. This is performed by the "traffic controller" components that are available on both sides (source and destination). The traffic controller on the destination end forwards the messages from client further to the source traffic control. Simultaneously, the destination traffic controller initializes a packet buffer wherein all the forwarded messages are stored. In the copy and restore phase, the snapshot procedure is initialized. The snapshot creates a checkpoint of the process is obtained, during which the service is not available (downtime) ensuring a consistent state snapshot. Afterwards, the migration is executed through the migration controllers available on both ends allowing a consistent state copy to be transmitted to the destination server and resumed by restoring the provided snapshot. The replay phase is then started. In this phase, the destination server attempts to catch up the source server state by replaying the packets starting at the checkpoint time stamp. Furthermore, the output of both servers are consistently compared through out the phase by the traffic controller. Finally, once the packet buffer is processed, i.e., empty, the switch phase takes place. In this phase, the traffic controller compares the outputs one last time. If the output show consistency, the destination server is then declared ready to take over and the corresponding network communication is thus readjusted. Figure 2.3 illustrated the explained procedure.

2.5 Components in Automation

A component represents an important aspect within the framework of the dissertation since it is the entity being modified or deployed. A component according to ISO/IEC 10746-2

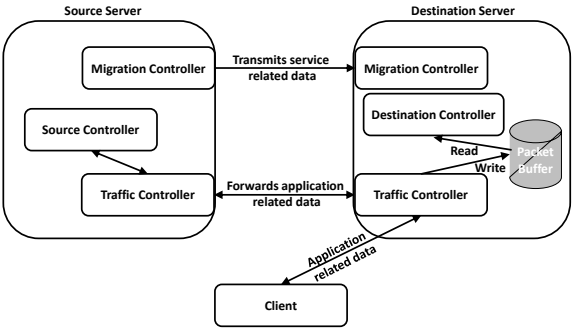


Figure 2.3: Redundancy Migration [42]

[73] is defined as "An object that encapsulates its own template, so that the template can be interrogated by interaction with the component. The template and other instantiation parameters are expressed in a form that allows them to be updated during the lifetime of any system of which the component is to form a part, allowing alternative realizations of the component to be substituted.". In the standard ISA-88 modes (standard ISA-88.00.01, 7.3 Table 1) for batch modes are presented, which can be used as operating mode for the components. A distinction is made between procedural units and units for resources. This roughly corresponds to the distinction between Single Control Units (SCU) and Group Control Units (GCU) similarly as introduced in [32] and [43].

2.5.1 Single Control Unit (SCU)

A SCU can be fundamentally seen as the software component that directly operates the device. For each device, there exists a SCU that contains the control logic which operates it. The SCU has an interface through which the states can be read and the commands can be submitted.

2.5.2 Group Control Unit (GCU)

A GCU plays the orchestrator role among the SCU components or parent group that coordinate other SCU components in the control hierarchy. The GCU can possibly be representing a physical entity in a plant so that it orchestrates the function of the devices with whom it gets in contact. The GCU mainly ensures a certain sequence of operations and preserves possible operation conflict through occupation handles which consequently could enhance the safety aspects.

2.5.3 Procedures

A procedure is defined as "a self-contained control unit that permits the automatic performance of an entire functionally coherent block of tasks" [70]. Procedures can be seen

as a list of instructions that are given to the GCUs or SCUs to perform a certain task or to execute a production process. Examples of a procedure can be:

- "Transport product to storage",
- "Pump reactants into tank" or
- "Navigate right, left, right, turn".

2.5.4 Inner Structure of a Process Control Component

Whether a component is a SCU or GCU, the inner structure of the component follow the scheme shown in Fig. 2.4 according to [43]. From top to bottom, the order passes through the following stages:

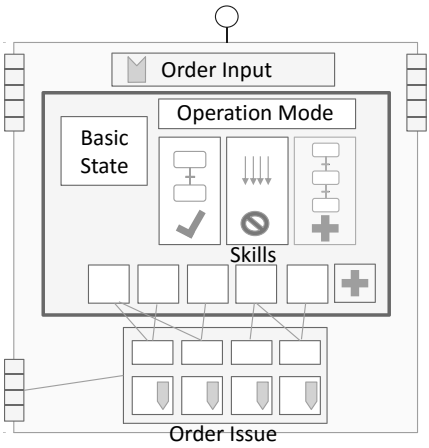


Figure 2.4: Component Inner Construction. The Operation Modes and Skills are illustrated

- Order input: Each component service interface has an order-input function. The task of the order receipt is the organization of the access authorization, the receipt and the acceptance and examination of orders. Accepted orders are transferred to the order execution for processing.
- Operation mode: The structure of the functionality of operational components is realized by means of operating modes. According to the designated component considered, the component is always exactly in one operation mode. In every operation mode, a component can be in different operating states. For example, in the case of a SCU, it parametrizes the skill of a motor, i.e., the skill "Move" which indicates the rotation of the motor armature. It can be parametrized to move right or left using the operation modes indicating the rotation in clockwise or counter clockwise respectively. The operation modes themselves are components or component type-specific

and therefore cannot be defined in general terms. The change between operation modes basically occurs only by passing through the basic state. If this is not the case, the change is then considered a special case and has to be explicitly defined for each desired change. The component state machine therefore contains the ground state “Basic State”.

- Skill (not to be confused with “capability”): The definition of a skill in the context of the component structure is taken as the the ability of the software component to execute a certain capability in an asset. Hence, an asset can have more capabilities than skills, i.e., each skills has a corresponding capability, however, each capability does not necessarily have a corresponding skill. A skill can be seen as the software representation that controls a capability in the information world.

2.5.5 Adaptation in Industrial Automation Systems

Adaptation in Industrial automation systems can occur on the different levels of the automation pyramid. For example, adaptation can refer to field devices adaptation (field level) or to control software adaptation (control level) or at the optimizations strategies (manufacturing execution system level).

On the field level, an assistance system is proposed in [9] and [10] that uses a resource-based view of automation systems and achieves a bottom up planning of logistic plants using a multi-agent system approach. The objective of the assistance system is to generate all possible solutions that can be used to realize an industrial automation system in the intra logistic layout engineering phase. A detailed planning of material handling system is presented in [11]. Furthermore, other approaches as in [35] and [75] propose a scheme to increase the reconfigurability of material flow systems. Moreover, an automated analysis scheme is shown in [5] that interlinks both layers the field and control layers. The objective of this automated analysis comes handy during the exchange of field devices. A common problem that arises is the incompatibility of the former devices control software and the newly introduced devices in the field. The aim of the automated analysis scheme is to identify the incompatibilities in the available software and hence adapts it.

The aforementioned contributions can be linked to the presented adaptation concept extending it to cover the hardware level and consider the interlinking parameters (e.g., material flow systems). However, in the scope of this dissertation, the adaptation focuses solely on the control level, i.e., control software adaptation through software redeployment, and does not consider further adaptation on other levels.

Furthermore, a model-based approach that utilizes design patterns is shown in [34] and [26]. The approach utilizes the software methodology of design patterns which considers the functional and non functional requirements for application deployment. The discussed method can be used in the presented work to form a list of constraints and optimizations criteria to be considered during the redeployment process.

2.6 Methodological Fundamentals - Graph Theory

In mathematics, a graph can be described using two elements: nodes and edges. A graph maybe undirected which indicates that there is no direction distinction when using an edge

between two nodes and vice versa for directed graphs (also called digraphs), or unweighted which indicates that the cost or the overhead of traversing between the nodes is not taken into consideration and vice versa for weighted graphs [96] [12]. Relevant glossary of graph theory [14] is included in this section:

2.6.1 Bipartite Graph

A bipartite graph (or bigraph) does not comprise an odd-length cycles. In bipartite graphs, the nodes can be divided into two disjoint sets that are independent from one another.

2.6.2 Adjacency Matrix

A square matrix that describes the connections between the nodes inside a graph. In undirected graphs, the adjacency matrix is symmetric.

2.6.3 Star Topology

A star topology indicates a tree graph with one internal node and $n - 1$ leaves.

2.6.4 Hub and Spoke Topology

A widely used topology in networks that is driven from the spoke-hub distribution paradigm that traffic planners used to optimize traffic transportation systems. The topology was later adopted in telecommunication and information technology sectors. The topology is also sometimes called star topology, however, it describes different star graphs where the internal nodes of the trees are connected in a certain form.

2.6.5 Mesh Topology

A full mesh topology indicates that an edge exists between any two nodes inside the graph. On the other hand, mesh topology indicates a fully connected graph, however in a random manner and a direct connection between any two nodes does not necessarily exist.

2.6.6 $K_n - K_n$ Topology

A topology with two copies of a full mesh network connected via a bridge.

2.6.7 Neighbor

An adjacent node to a given node, i.e., directly connected to it.

2.6.8 Valency

denotes the degree of an entity, i.e., the number of incident edges. Valency of a node indicates the incident edges of the node.

2.7 Methodology - Load Distribution

As mentioned previously, the adaptation system perform a 2-stage optimization. In the second optimization, one of the main objectives is to distribute the load among the available computation resources. The topic of load distribution has been discussed from different perspectives. In the following subsections, the different perspectives are clarified:

2.7.1 Load Distribution Algorithms

Various domains have presented a conceptual formulation for the load balancing problem as the problem exists in different field disciplines. The fluid particle approach presented in [47] describes how mapping tasks to processor nodes using the physical analogy. The analogy shows how the fluid particles arrange themselves over a flat container considering their different viscosity and the acting forces. This analogy is used to model the affecting factors in the load balancing problem.
In informatics domain, various swarm algorithms use ants as autonomous agents to discover the network and balance the load among the available nodes in the network [64] [16]. An advantage presented by these methods is the decentral behavior presented by the Peer-to-Peer (P2P) systems. The ant agents wander around in the network through P2P communication collecting information about resources and available loads in the nodes. Further modifications to the algorithm to include stigmergy (i.e., pheromone communication) and particle optimizations are presented in [62].

2.7.2 Multi-Core Processing Analysis

Dynamic load balancing in multiprocessor is studied using a diffusion scheme in [22]. The objective of the load balancing in this domain is to distribute the load such that each processor receives and performs the same amount of work. The analysis presented here includes no a-priori estimate of load distribution. It is assumed that all tasks are independent, thus not influencing the execution order nor the executing processor. The loads are composed of continuous load units and each single task can only be performed by a single processor. Moreover, it is assumed that the temporary induced time and communication costs for the load transfer is significantly smaller than the task execution cost which imposes some limitations in the general applicability sense.

The general description of the load can be described by the diffusion model in Eq. 2.1.

$$x_i^{(t+1)} = x_i^{(t)} + \sum_j \alpha_{ij} \left(x_j^{(t)} - x_i^{(t)} \right) + \eta_i^{(t+1)} - c \tag{2.1}$$

Where x_i^t quantifying the work distribution at time t at node i. The middle term in the right hand side describes the load exchange between the nodes, where α_{ij} are the non negative constants that describe the fraction of loads that are exchanged with other nodes. The convention dictates that $\alpha_{ij} = 0$ in case the nodes are not connected. $\eta_i^{(t+1)}$ describes the newly added task at each processor at time k for processor i and c is a constant value that describes the amount of processed task at each node. Considering the homogeneous problem, it can be assumed that:

$$\psi_i = \eta_i^{(t+1)} - c = 0 \tag{2.2}$$

Tuning α_{ij} can enhance the performance and more importantly affect the load transfer stability. Rewriting Eq. (5.2) as

$$x_i^{(t+1)} = \big(1 - \sum_{j=1}^{n} \alpha_{ij}\big) x_i^{(t)} + \sum_{j=1}^{n} \alpha_{ij} x_j^{(t)} \tag{2.3}$$

yields a transition matrix A with $a_{ii} = 1 - \sum_{j=1}^{n} \alpha_{ij},\ 1 \leq i \leq n$, and $a_{ij} = \alpha_{ij}$ otherwise. Under a natural boundary condition $\sum_j \alpha_{ij} \leq 1$, A induces a Markov chain

$$x^{(t+1)} = Ax^{(t)} \tag{2.4}$$

with

$$A = \begin{bmatrix} 1 - \sum_j \alpha_{1j} & \alpha_{12} & \dots & \alpha_{1n} \\ \alpha_{21} & 1 - \sum_j \alpha_{2j} & \dots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1} & \alpha_{2n} & \dots & 1 - \sum_j \alpha_{nj} \end{bmatrix} \tag{2.5}$$

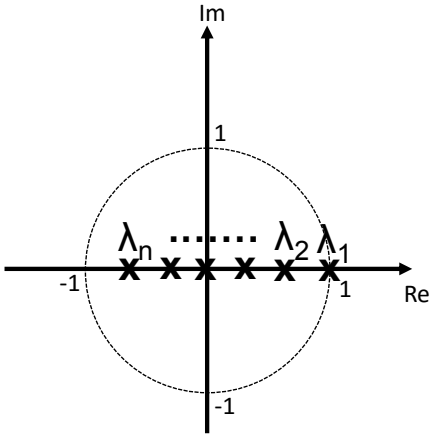


Figure 2.5: Poles demonstrated on the Unit Circle

According to Perron-Frobenius theory [49], since A is a double stochastic and an irreducible matrix, computing the eigenvalues of matrix A will result in

$$-1 < \lambda_n \leq ... \leq \lambda_2 < \lambda_1 = 1 \tag{2.6}$$

Using Eq. 2.6, it can be clearly seen in Fig. 2.5 that the second Largest Eigenvalue Magnitude (SLEM) determines the rate of convergence of A [22]:

$$\gamma(A) = \max\{|\lambda_2|, |\lambda_n|\},$$

where λ_i are the eigenvalues of A

Defining Γ_i as

$$\Gamma_i = \sum_{j=1}^{n} \alpha_{ij} \tag{2.7}$$

In case λ_n is equal to -1, the system does not converge which can be avoided if one of the following conditions holds (provided that the network is a connected graph):

- $1 - \Gamma_i > 0$
- the network is not a bipartite graph
- both

To compute the fastest convergence transition matrix:

$$A_M = (A_A + gI_n)/(1 + g) \tag{2.8}$$

where $A_M = (a_{ij})$

$$g = \min\left(-\frac{(\lambda_2 + \lambda_n)}{2}, -\min_j a_{jj}\right) \tag{2.9}$$

With A_M and A_A being the linearly modified optimum feedback system matrix and an arbitrary initial system matrix respectively. The scalar multiple g manipulates the eigenvalues and adjusts the feedback system matrix accordingly using Eq. (2.8). The cited method ensures an upper bound to the convergence time using the spectral gap by minimizing the second largest pole λ_2 (since the largest pole is always equal to 1) to the minimum possible value, the total settling time is reduced. This method provides an intuitive way for finding the least value for the second dominant pole. Further matrix manipulations are extensively shown in [89].

2.7.3 Consensus Networks

Fast consensus in Markov processes and load balancing in networks are shown in a framework of theoretical analysis in [67]. The equations shown in this contribution provide the fundamentals for modeling load balancing in the dissertation. Furthermore, stability, convergence and system dynamics analyses are also presented.

2.7.4 Semi-Definite Programming

The minimization of the SLEM can be formulated as a semidefinite program (SDP) as shown in [13] and [56]:

$$\begin{aligned} \text{minimize} \quad & \beta \\ \text{subject to} \quad & -\beta I \preceq A - \frac{1}{n}\mathbb{1}\mathbb{1}^T \preceq \beta I, \\ & A_{ij} \geq 0,\ A = A^T,\ A\mathbb{1} = \mathbb{1}. \end{aligned} \tag{2.10}$$

With:
$A\mathbb{1} = \mathbb{1}$ indicates the stochasticity of the Matrix.
A describes a Markov chain X_t over finite state space S and cardinality $\mathbf{S}$ (which in this

case is the number of columns of A).
$\mathbb{1}$ is vector of ones of dimension $\mathbf{S}$.

Semidefinite program is a standard type of convex optimization problems, for which various approximation algorithms have been developed. Most of them are based on the interior-point method. The objective defined in the optimization problem is to find an optimal solution for edges weights that perform the fastest mixing time in a provided hub spoke network. In [14], it is shown that for a given symmetrical network, there exists an optimal solution that contains same symmetry, i.e., a fixed-point of the automorphism group of the underlying graph defines a set of edge weights that carry same weights. Thus, restricting the optimization problem to consider only the distribution matrices with equal parameters for symmetric edges which significantly reduces the number of free parameters in the optimization. Similar examples are shown to solve the problem in Eq. (2.10) for some types of topologies that contains high level of symmetry, e.g., ring (cycle), wheel (star) and $K_n - K_n$ [14]. The SDPT3 solver is used with YALMIP MATLAB toolbox to compute the regression model. The former implements a primal-dual infeasible-interior-point algorithm and shows robustness for medium-size semidefinite optimization problems [88].

2.7.5 MATLAB-YALMIP

YALMIP is a MATLAB toolbox [60] that facilitates solving semidefinite programming (SDP) and linear matrix inequalities (LMI) problems through providing an interface to external solvers. The tool can be used to model the problems hence the name YALMIP- "Yet Another LMI Parser". YALMIP uses MATLAB commands. The toolbox is utilized in the scope of this work to solve SDP problem of rapid mixing of hub and spoke networks.

2.7.6 Simulated Annealing

While performing redeployment of load packets, finding the optimum distribution according to the defined optimization/constraints criteria is the objective, e.g., maximize the reserve at each computation point. Enhancements to the algorithm that performs the redeployment can be inspired by the Simulated Annealing (SA) probabilistic optimization technique [55]. SA can be utilized to solve combinatorial optimizations which is useful in the scenario of distributing discrete load packets with heterogeneous sizes. The SA methodology shows a superior performance to the gradient descent method, as it helps escape from local optimums to find a global optimum.

2.7.7 Optimal Distribution Solvers - Z3 SMT Solver

Satisfiability Modulo Theories (SMT) problem is a type of decision problems that can utilize a constraint satisfaction problem concept to conduct an optimization. Z3 is a SMT solver introduced by Microsoft Research [23]. The solver targets solving software analysis problems. In [81], Z3 solver is used to compute valid configurations that utilizes SMT-based constraint resolutions. The approach configures IEC 61499 [91] systems for deployment by calculating a plan that considers the available resources and feeds it as an input to the solver. An on-going work is presented in [85] which aims to solve architectural optimization

in the design phase (offline). The solutions presented by the aforementioned contributions present basis for the usage of SMT solvers to compute an optimal deployment profile that considers the available resources and the application's overhead.

2.7.8 Deployment in Automotive Open System Architecture (AUTOSAR)

In the Automotive Open System Architecture (AUTOSAR), during the system configuration phase, a similar problem is faced, where mapping of software components on control units poses an important requirement. In [69], an approach is presented which inspires the adaption concept in this dissertation. Firstly, a decomposition is performed such that all compound components are broken down to atomic components. These components are further classified according to their hardware dependencies. On the one hand, the hardware dependent components are deployed directly to the required devices. On the other hand, an optimization using evolutionary algorithms is used to distribute the rest of the components optimally. The optimization criteria and constraints utilized are:

- resource requirements,
- real time scheduling,
- and minimization of the intercommunication overhead between the control units.

2.8 Agents Systems

A further approach that is utilized as the drive for load distribution is the agents systems approach. According to [39], agents systems can be classified as shown in Fig. 2.6.

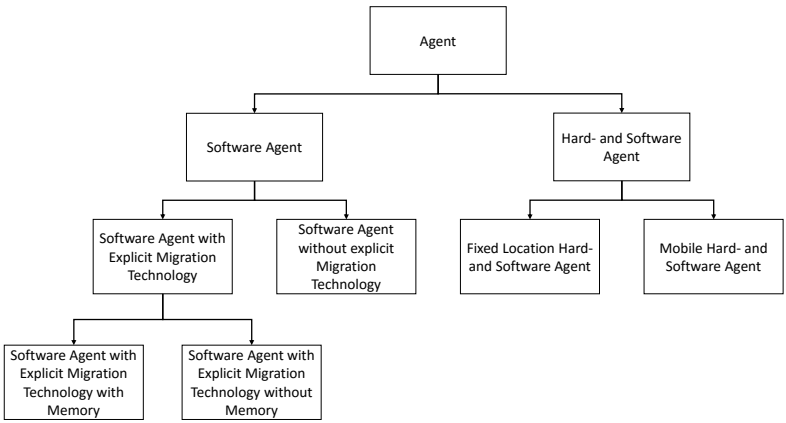


Figure 2.6: Agents Classification [39]

2.9 Market-based Multi-Agent-System Approach

Load Distribution has been investigated from an agent based approach in the domain of smart grids [58] and decentralized power and grid control [59]. Undesired system responses that can arise in runtime have been pointed out in similar situations as shown in [58]. Thus a stability and convergence analysis to investigate system response (e.g., load oscillations) must be conducted. In [59], a framework is presented that simulates an event-driven marketplace which is regulated by agents for decentral power and grid control. The proposed approach aims to monitor demand and supply and achieve balance through communicating with external marketplaces via an agent broker. The concept of agents that collect data and can foresee dynamic changes inspire the presented work. Furthermore, the marketplace auction mechanisms presented in [38] provide an insight to a realization of the multi-dimensional resource optimization (albeit between rival network participants) that is presented in this dissertation.

2.10 Agents Systems Hierarchy in Automation

Agents systems can be used to realize the different control components, i.e., procedures, GCUs and SCUs. As mentioned in Sec. 2.5, the control components form the fundamental unit in redeployment. Taking the perspective of agents system can provide an insight to develop an approach or a methodology into how the load balancing can be performed. As shown in Fig. 2.7, there exists a life-cycle for each procedure from initialization passing through execution and accomplishment till the deletion stage. During these stages, the production execution takes place, thus dynamically changing the system conditions [39] [31].

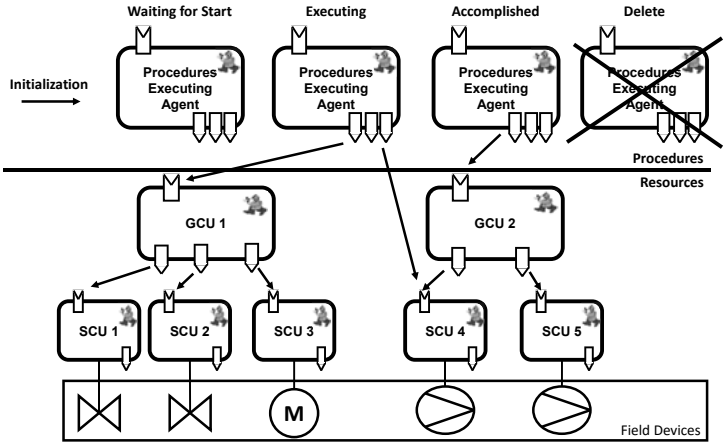


Figure 2.7: Life-Cycle of a Procedure Agent [33]

2.11 Agent-Based Planning of Production Sequences

Agent-based planning is utilized in the second scenario of the prototype implementation. An approach to compute solutions to realize product recipes is shown in [76] which can offer a profound insight into how to extend the resource based agent approach demonstrated.

2.12 Recipes Definitions

IEC 61512-1 [20] and ISA 88 [52] aim to standardize the used terms and concepts that describe recipe based operations in industry. The definitions of the relevant terms that will be used in this dissertation are defined in [52] which includes: “Modes of operation”, “Recipe”, “Master recipe”, “Control Recipe” and “Process”. Figure 2.8 show the flow of the different procedure recipes according to the standard.

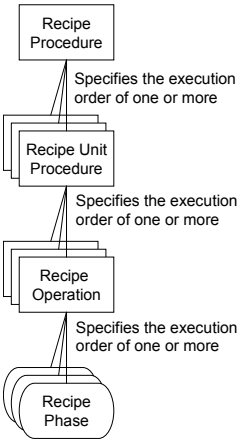


Figure 2.8: Procedures Recipes according to ANSI/ISA-88 [52]

2.13 Tools - Discovery

Multicast Domain Name Servers (mDNS) Discovery is a decentral mechanism that is used to explore the network topology in the implementation. In the absence of a conventional unicast DNS (uDNS) Server, Multicast DNS protocol enables performing DNS-like operations. A zero-configuration service that provides IP networking by resolving host names to IP addresses. Apple Bonjour and open source Avahi software packages are well known for their implementation using mDNS. The protocol is published as RFC 6762 [18]

2.13.1 Bonjour Protocol

Bonjour protocol (formerly known as Rendezvous) is introduced by Apple for the objective of automatic discovery. The protocol uses mDNS service records to discover devices, i.e., computers, printers or offered services in the local area network. The protocol can operate under the following operating systems: Mac OS 9, macOS, Linux, Berkeley Software Distribution, Solaris, VxWorks, and Windows [51].

2.13.2 Mechanism of Operation

An mDNS client resolves host-names through sending an IP multicast query message. This message requests the corresponding target host to identify itself through a multicast response message which encloses its IP address. Machines found in this subnet can also update their mDNS caches with the response information. Alternatively, the relinquishment of a claimed domain name can be performed in the same manner, however with a multicast response with a time to live (TTL) equal to zero.

The same programming interface and packet formats as the uDNS are utilized. The multicasted message is basically a User Data Protocol (UDP) packet with a similar payload that comprises a header and the data. The packet is addressed to the following destinations:

- In Ethernet: the standard multicast MAC address 01:00:5E:00:00:FB (for IPv4) or 33:33:00:00:00:FB (for IPv6).
- IPv4 address 224.0.0.251 or IPv6 address ff02::fb.
- UDP port 5353

mDNS does not provide information about the type of the device found or its status. Utilizing the DNS Service Discovery (specified in RFC 6763 [17]) can provide an insight into the device nature and its status. However, this is not included in the scope of this work, as it is assumed that all devices inside the network are automation exclusive devices that can operate and execute process control functionalities.

2.13.3 Reconfiguration of Real-Time Fieldbus

Device outage as well as newly introduced devices in the network require the dynamic reconfiguration of the Fieldbus. The work presented in [74] provides a foundational groundwork to the presented adaptation concept which can extend its autonomy in terms of adapting to disruptions in the shop floor.

3 Runtime Adaptation Concept

In this chapter, the concept of runtime adaptation is explained. As previously discussed (cf. Sec 1), with a highly dynamic environment in I40, an adaptation concept that distributes the loads and optimizes the resources usage is required. The described runtime adaption system comprises three subsystems as illustrated in the UML diagram in Fig. 3.1:

- Function generation and Engineering systems
- Deployer system (Executor)
- Redeployment management/optimization system

The function generation engineering systems are used to create control logic and perform system engineering in the initial deployment phase. Once the control logic components are ready, they can be deployed via the deployer system. The deployer system acts as an interface to the computation nodes where software components can be deployed, redeployed or copied for redundancy generation. The redeployment system uses the deployer system as a tool to perform load balancing on and resource optimization to the computation nodes. In the scope of the presented concept, only the redeployment and the deployer systems are considered.

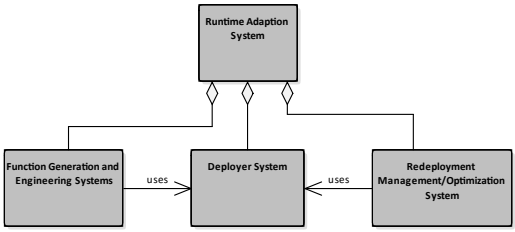


Figure 3.1: Proposed System Components

In order to explain the system objectives and how the system operates, the used terminology must be first defined.

3.1 Terminology and Definitions

In the framework of the adaption concept, various entities in the plant are involved. This section aims to define them and discuss the assigned conventions.

Firstly, the time frame, to which the term “runtime adaptation system” refers, is defined as the runtime of the whole production starting at the function generation and engineering

stage and finishing at the online stage during the production where the redeployment takes place.

Throughout the dissertation, the graph notation is used to represent the computation networks. The term “node” is used to describe a computation node, unless explicitly stated otherwise. The term “edge” describes a network connection between two nodes. The word “device” is used to describe field devices in the shop floor.

The term “component” indicates a software component which is considered as the atomic unit for the deployment and the redeployment processes.

The deployment process is the act of initial allocation of software components on the control nodes (whether PLC or other available computation nodes/resources), while redeployment refers to the rearrangement through copying or reallocation by moving the already deployed software components to other destinations in the network.

The adaption system operation can be classified into two main phases: initial deployment and online redeployment. The objective of the former is to automate the initial deployment phase. This is done through the deployment of the required procedures, SCUs and GUCs in the planning phase. The objective of the latter is to optimally prepare the operation of control systems for the production and possible faulty situations. This is usually done through cyclic execution/monitoring to maintain certain qualities in the network and achieve a better utilization of resources, e.g., availability, redundancy, communication Quality of Service (QoS), etc.

3.2 Concept Overview

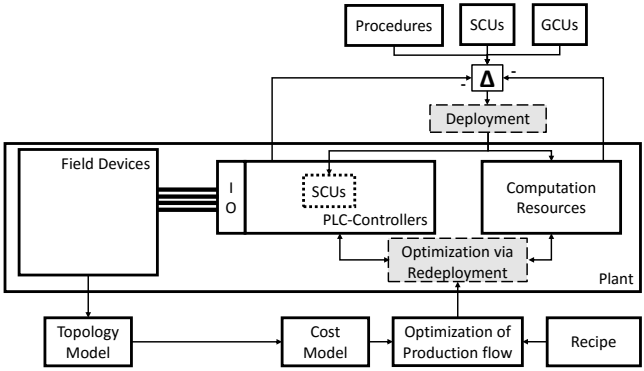


Figure 3.2: Concept Information Flow mapped on the Architecture

The Concept is illustrated in Fig. 3.2. The two gray colored boxes indicate the main contribution areas of this dissertation. Table 3.1 clarifies their objectives.

The concept introduces a deployment platform that can be used to deploy the different components to the computation nodes in the plant. Initially procedures, SCUs and GCUs are deployed to the computation nodes using the deployment interface. The computation

nodes considered in the concept are PLC-controllers or further computation resources like electronic boards or Industrial PCs (IPCs). On the PLC controllers, the SCUs are marked with a dotted line box to indicate their fixed nature in terms of component placement. By definition, SCUs require real time communication with the field devices since they read and write signals directly from and onto the devices [74]. This constrains the network wiring plan, i.e., PLC IOs coupling with the field devices, to be fixed and pre-known. Under those circumstances, the initial deployment is performed where the required SCUs are deployed to their respective PLCs to ensure that the realtime requirements are met. The deployment is done according to a delta model. The objective of the delta model is to compare and indicate the current missing or unnecessary available components and consequently deploy or delete the components respectively. On the other hand, the deployment of the procedures and the GCUs can be flexibly deployed and redeployed according to an optimized distribution profile in the network. Initially an offline deployment plan of the procedures and GCUs can be optimized using a solver [23]. Important to realize, such optimizations require a considerable time to execute since they often perform brute force optimizations. Such time consuming optimizations do not provide a suitable solution to perform on the fly redeployment of the software components during runtime. Thus requiring heuristic optimization via redeployment algorithms. The redeployment stage ensures an optimal distribution of the GCUs considering the available resources as well as it ensures a robustness in the execution by continuously monitoring the nodes and redistributing the GCUs to secure nodes, e.g., by ensuring redundancy or placing GCUs at nodes that are not planned for maintenance or an expected outage. However, depending on the use-case, optimizations can be performed according to different objectives and considering different constraints.

For example, the redistribution can be performed to ensure a balanced load distribution amongst the nodes in the network. Alternatively, the redistribution can be done to ensure an optimal placement of the software components in the network. Both perspectives provide different (sometimes opposing) objectives. For example, a load balancing objective can be to evenly distribute the load amongst the nodes, whereas a component placement objective can be to ensure the highest resources reserve at the placing location.

In specific use-cases, other optimization criteria can be derived externally, e.g., from the recipe. The optimization of the production flow can provide essential information to the redeployment strategy, hence, as shown on Fig. 3.2, an arrow is drawn from the production flow optimization to the redeployment optimization to indicate possible flow of information.

Table 3.1: Overview of the Main Focus of the Work presented in this Dissertation

	Description	Optimization Type
Deployment	Initial deployment of the required Procedures, SCUs and GCUs	-
Redeployment	Rearrangement -by copying or moving- of components on computation resources	Heuristic

The concept is analogous to a control loop as demonstrated in Fig. 3.3. The adaptation system comprises components that provide functionalities like sensing (measure), control

and actuate the applications in terms of resources consumption. The redeployment system is an add-on tool for automation that can enhance the performance of systems that consider adaptability. Important to realize, a static computation system (that has a static network plan and fixed resources or that is operated on a single computation point that has enough/redundant resources) does not require a redeployment system. The following sections are structured according to the components of a control loop and uses its analogy to explain the concept and clarify the system architecture and the functionalities provided by the different components.

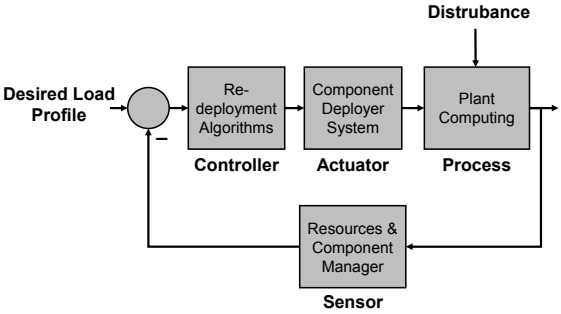


Figure 3.3: System Control Loop

3.3 Process

In the analogy, the process represents the computation operations performed in the plant. In order to perform these operations, the redeployed unit which influences the processes must be identified. A redeployment process describes the displacement or the copying of a software component which is represented in a function block or a control chart from one computation point to another. The redeployment unit varies according to the use-case and the system design. In this section, a classification of the different software component types are presented. The classification coarsely comprises two main classes, direct component deployment and container deployment.

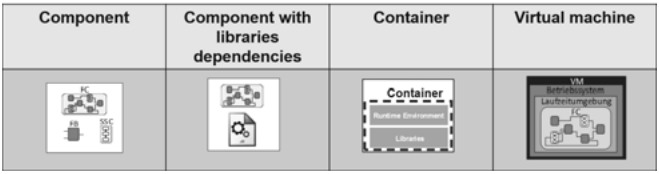


Figure 3.4: Examples of the different deployment platforms

3.3.1 Component redeployment

A component redeployment indicates a non-packed transfer or copying process of a component, i.e., without the use of an intermediate technology, e.g., containers from one computation point to another. In other words, the deployment is done within the Runtime Environment (RTE) using its legacy tools

Direct component redeployment

The simplest deployment platform is the direct deployment of components. This can be seen as transferring/copying a file from one node to another. On the one hand, in this scenario, it is assumed that both nodes have the required dependencies to read, execute and use the file.

Component with libraries redeployment

On the other hand, if this is not the case, the dependencies must be also transferred to the target. Usually the dependencies are found in library files or in a script form depending on the RTE. In this scenario, if the library files are operating system dependent (e.g., .dll or .so), the corresponding library files must be transferred to the right directory and in some cases must be included in the configuration files of the RTE.

3.3.2 Container Redeployment

An encapsulating container can be used to pack all the required dependencies. The objective of this container is to provide portability to systems through a framework that allows the transfer of components by encapsulating all their required dependencies in a container. This container can then be uploaded to a cloud where the container administrator of other nodes can download it. In the virtual machine example, library dependencies as well as the operating system are packed to ensure compatibility. This method can be costly in terms of resources consumption. On the other hand, in the docker container example, libraries are packed with the component to ensure compatibility without packing an image of the underlying operating system. However, it must then be ensured that the container is deployed on a similar or a compatible operating system.

3.4 Controller

The controllers describe the algorithms or the solvers that consider the optimization criteria and constraints in the load adaption system. The optimization criteria and constraints are discussed in Sec. 3.4.1 and shown in the UML diagram in Fig. 3.6.

Redeployment requires a logic to drive the process. Due to realtime constraints that can be imposed by hardware or software, the logic is based on heuristic optimizations and algorithms that can operate in runtime. Examples of such algorithms include:

- Load balancing algorithm: The load balancing algorithm ensures an optimal use of the resources provided in the network by moving the software components to different nodes according to the assigned constraints and optimization criteria (cf. 3.4.1).

- Component redundancy algorithm: The redundancy algorithm ensures a given degree of redundancy of components to provide a backup solution to node outage situations.

- Variants manager and load adapter algorithm: For the sake of completeness, the variants manager and load adapter are included as further algorithms that can affect the network load, however, they are not included in the scope of this research. A component can exist in different forms or "variants". Each variant offers a different quality of service and has a different resource consumption profile. Toggling between variants can be used to manipulate the consumed resources as well as the offered quality by the component [92]. On the other hand, a component can offer additive functionalities that can be considered as "luxurious" additions. Using the load adapter, switching on and off of these luxurious functionalities enhances the performance and reduces the overhead of a component respectively [44].

The decentral redeployment system concept introduced in this dissertation is demonstrated using the UML diagram shown in Fig. 3.5.

3.4.1 Optimization Criteria and Constraints

The following list shows the different optimization criteria and constraints that can be considered by the controllers. A UML illustration of the criteria and constraints is shown in Fig. 3.6.

- Controller to plant topology: an existing (or a possible) communication channel between the target node and the device controlled. This communication channel should support the minimum needed quality of service as well as the network protocol required by the application. However, it is assumed that the controller plant topology is hardwired and does not change. The network plan, quality of service and the corresponding protocol of this topology are provided as an information model.

- Controller (Computation Nodes): the utilization of resources, i.e., the computation, communication or storage overhead, on the controllers can be optimized through load balancing. Additionally, it should be ensured that the target is not scheduled to go offline in the upcoming specified interval of time.

- Controller to controller topology: In this point, there exist an optimization criterion as well as a constraint:
  - Constraint: An existing (or a possible) communication channel between the source node and the target node to ensure a smooth synchronization of the available states in the deployed component (this point is discussed further in the following section). For components with real time criticality, a corresponding QoS of service should be met in the communication channel between the source and target nodes. In case of peer to peer deployment, the same channel can be used to deploy the components otherwise, an alternative cloud solution can be used to deploy an image from the cloud.
  - Optimization criterion: The communication channel between the source and target node should be ideally a direct channel (i.e., without network hops) to reduce the temporary network communication overhead due to deployment.

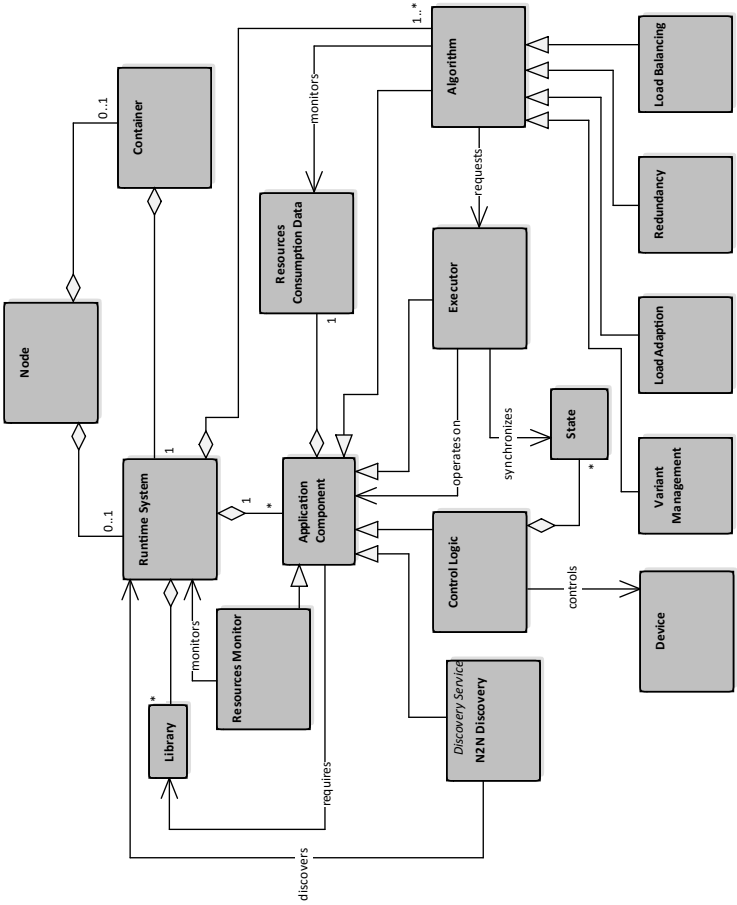


Figure 3.5: UML Diagram of the Redeployment System

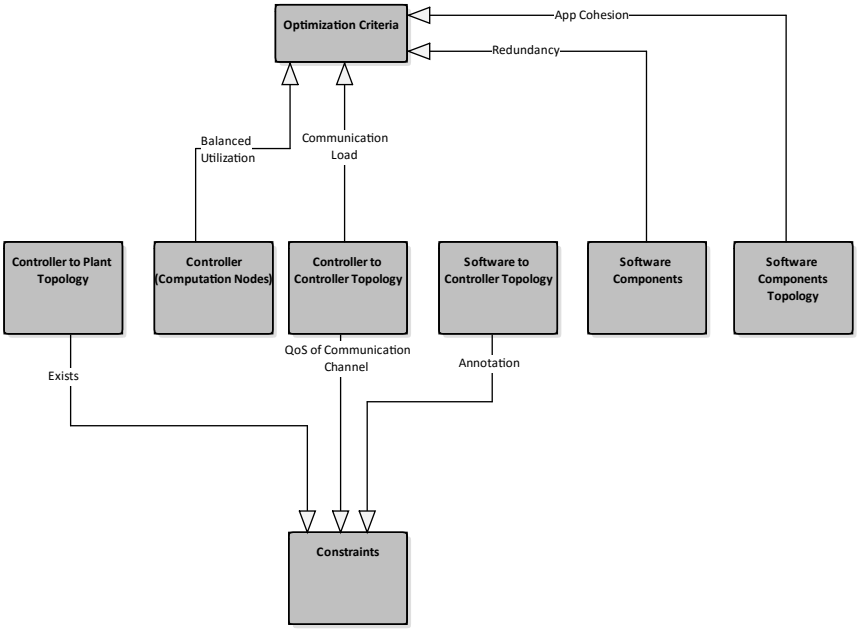


Figure 3.6: Illustration of the Optimization Criteria and Constraints

- Software to controller topology: The target node should have the necessary hardware to operate the deployed application and fulfill the software annotations, e.g., processor architecture, processing capacity, etc.
- Software Components: Ensuring a specific degree of redundancy of the software components.
- Software components topology: In order to avoid causing a communication overhead, the application cohesion should be considered, i.e., the application dependencies between the different software components should be taken into consideration. Ideally, a component should have no dependencies on other components found in other nodes.

3.4.2 Boundary Conditions

In the decision phase of picking the source and destination nodes that will perform the deployment, the following boundary conditions must be considered [29]:

- In the source and destination nodes, sufficient resources must be available for the transmission process itself.
- The destination node must have the resources and quality of services required to run the deployed software unit (communication, computation, memory, hardware, etc.)
- It must be ensured that the switching process between the new and the old component does not violate the real-time requirements.

In order to discover the neighbors and their resources profiles, the discovery profiles are defined in Sec. 3.6.1.

3.4.3 Stability, Performance Analysis and Performance Enhancement

Redeploying components in a network is a dynamic process which requires to be analyzed to ensure stability and convergence. Moreover, different indicators should be available to measure the performance of the algorithms. Thus a dynamic model is constructed using the state space representation. The model utilizes a discrete time domain. Chapter 5 is dedicated to discuss this topic due to its protracted nature.

3.5 Actuator

The actuator in the adaption system can be defined as a tool that can influence the load in the network, i.e., the nodes or the communication channels. The load can be influenced through moving or copying the components in the network.

3.5.1 Load Distribution Executor

The load distribution mechanism varies according to the considered deployment platform (cf. Sec. 3.3). The deployment of a container or a component with library dependencies are included in the scope of the concept.

Component and Library Deployment

Fig. 3.7 illustrates the logic flow of the load distributor in the case of a component with library deployment. After a decision is taken to deploy a component, the component is serialized and the library dependencies are listed. On the one hand, if the required libraries are available at the target node, the executor makes sure that they are loaded and the serialized data is then transferred and then instantiated (by de-serializing then creating the component at the target node). On the other hand, if the libraries are not available at the target node, if the target and source nodes have the same Operating System type and compatible architectures (e.g., if they both use Windows or Linux as an OS, then a .dll or .so file will be sent respectively), otherwise a gateway server must exist from which the libraries can be requested and consequently downloaded. A synchronization must be performed in case the deployed component has states.

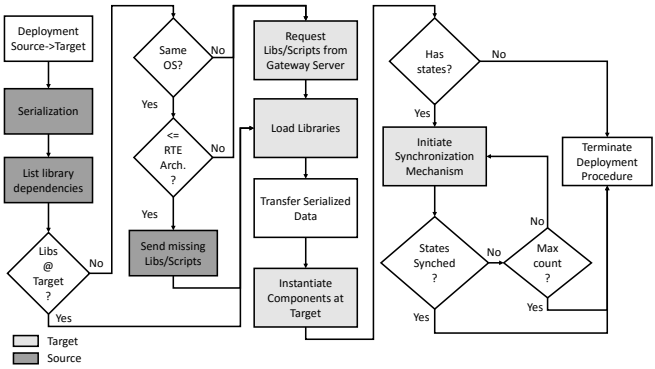


Figure 3.7: Logic Flow of the Load Distribution Executor

Container Deployment

Deploying a container requires the pre-installation of a container daemon. This daemon provides an interface to download container images from the cloud. After the deployment of a container, the states must be synchronized. Containers provide a ready solution to the deployment. Hence, the container solution is described in this section as an alternative solution. Nevertheless, the container solution is not considered in the scope of the implementation as ensuring a certain realtime capabilities using a container based approach can impose a problem. For example, in Docker containers, the internal mechanism of container operation and memory management might not be suitable (nonetheless showing promising results) in certain automation application [40].

Synchronization

In order to fully accomplish a deployment process, the deployed component must be initiated and in case of a redeployment, its states (if there exists any) must be synchronized.

The different synchronization strategies are shown in Sec. 2.4. The performed communication whether sending or receiving message from or to the client should be regulated through a traffic controller as shown in Fig. 3.8. Thus the deployed components are iterated and the communication interfaces are detected and consequently forwarded. Using the shown architecture, the message forwarding mechanism can be utilized to implement a proxy like communication. When a message is sent from the Source (SRC) client, the traffic controller packs the outgoing message in a further message enclosing the original address and forwards it to the Destination (DST) client. The DST client then writes the message in the packet buffer and forwards the message further using the enclosed address. When a read request is initiated, the traffic controller on the SRC client side requests the DST client to read the desired data and similarly it writes a copy in the packet buffer. A direct read request can be performed directly, however, with the mentioned mechanism, a deterministic synchronization can be guaranteed. Moreover, the traffic controller on the DST client side records the timestamps of the history in the packet buffer. Including the timestamps is essential in the replay phase to simulate the same timing of execution which can be crucial in some logics, e.g., a delay timer.

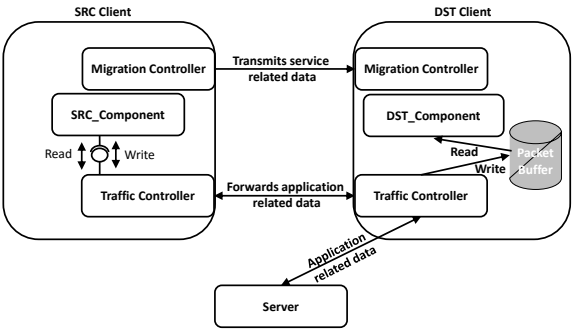


Figure 3.8: Redundancy Migration

3.6 Sensor

3.6.1 Resources & Component Manifestation

In order to compare if a process can be carried out on a certain node, information about both the demanded resources as well as supplied resources must be available.

Resources reserve level

At each computation node, a manifestation of the resources available should exist. The manifestation should demonstrate the utilization and availability of the following:

- CPU

- Dynamic memory
- Storage
- Communication capacity (bandwidth and data-transfer rate)
- Network adapter

Additionally, a discovery profile to show the nodes and devices in the vicinity of each node. There are two types of topologies that should be defined in the context of deployment:

- Node to Node (N2N): In N2N topology, a decentral technique is used where a computation node can discover other computation nodes that are directly connected to it or indirectly through its neighbors. An end point of a node can be reached through one or more network infrastructures. For example, in Fig. 3.9 all the nodes can reach each other through Ethernet communication. Additionally, Node 5, Node 1 and the smart device are connected via a Fieldbus, however, the computation nodes have a master role and the smart device has a slave role in the latter network which can be used as a criteria to differentiate between the node types.
- Node to Device (N2D): It is assumed that the N2D topology is fixed and is provided to the deployment as a given information model. The topology is hardwired and does not change throughout the execution time.

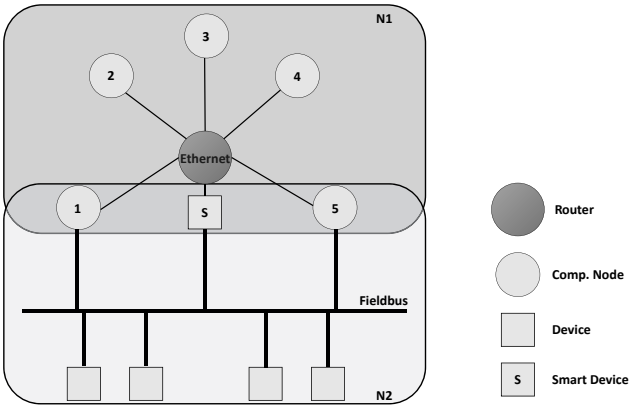


Figure 3.9: An Example of a Typical Industrial Network

Application Consumption Level

It is assumed that all applications utilized in the context of deployment are evaluated in the sense of resource consumption. Each application must enclose a profile that demonstrates the resources consumption of the best and worst case execution.

3.7 Disturbance

This section describes the disturbance analogy that occurs in the plant. A possible disturbance in the plant computing process can be caused by the introduction or the termination of resources or load, i.e., computation node or an application respectively. Furthermore, the change of the boundary conditions of the hardware in the computation node or the consumption of a software component can also cause a disturbance in the process, e.g., addition or elimination of a provided network interface in a node, modification of the desired redundancy, etc.

3.8 Architecture Overview

The described concept can be mapped using a central or a decentral architecture as shown in 3.10. The architecture shows both architectures for centralized and distributed architectures in the upper and lower frames respectively. The figure shows deployment using container technology that downloads an image from a repository located in the cloud where all containers are precompiled. The figure demonstrates the different information models with their contents, the information flow indicated by the arrows, the monitoring components, the executer components and the communication interfaces available in the architecture. This concept has been developed in the framework of a collaboration with ABB [45]. The central reconfiguration is triggered using a manual input from a human while the distributed (decentral) reconfiguration is automatically triggered by thresholds that activate different algorithms. Important to realize that each architecture has its own advantages and drawbacks. A central architecture outperforms the distributed one in terms of distribution profile computation, i.e., a central architecture computes an optimal solution for the load distribution while a decentral architecture uses heuristic approaches. On the other hand, a decentral architecture surpasses the central one in terms of readiness for dynamic disturbances and adaptability, since the decentral approach solely relies on partial information models and does not require global information to be known. In the middle of the figure, two devices are illustrated, along with their internal containers, their states, and the information flow from both devices to the different architecture types.

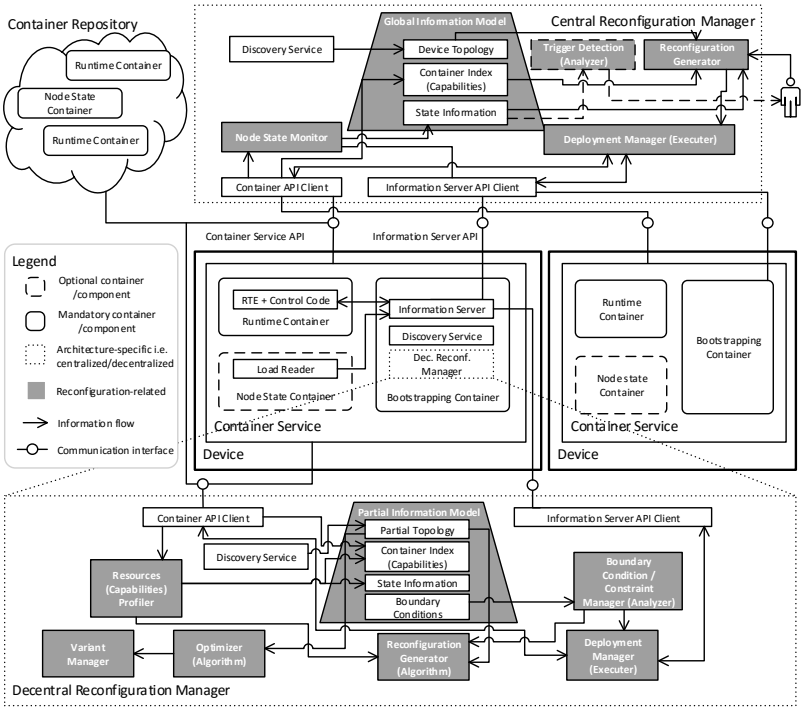


Figure 3.10: Central vs. Distributed Architectures

4 Modeling Fundamentals

In the framework of this dissertation, two models are constructed to tackle and analyze the load balancing problem from two prespectives, a linear approach and a non-linear approach. The two models share mostly the same fundamentals, however, some characteristics are modeled differently which are highlighted in this chapter. Furthermore, the considered assumptions in both models are discussed. Table 4.1 demonstrates the different perspectives and compares the models regarding the load, transfer and the decision taking aspects.

4.1 The Load Balancing Model

The considered model in the scope of the dissertation is classified into the following aspects:

4.1.1 Network Model

Firstly, the common characteristics are discussed. The networks are modeled using the graph notation, i.e., nodes and edges to represent computation points and communication paths respectively. An edge can only connect two different nodes, i.e., a self connected node edge is not applicable as it has no semantic meaning in the presented model. Edges are considered to be bidirectional and unweighted edges, i.e., the communication edges are undirected and allows flow in both directions and the communication load is considered negligible. Only connected graphs are considered, i.e., all nodes can be reached and belong to a network. All decisions and communications are performed decentrally. A node can only communicate with other nodes and retrieve information only within the node's neighborhood. Load transfers are executed in a synchronous manner. In both approaches, discrete time domain is considered.

Depending on the model, whether it is linear or non-linear, in the respective order, the following aspects vary:

- The network sizes can be static or dynamic
- Load balancing optimization process occurs in a central or a decentral manner

4.1.2 Load Model

Only nodes can carry loads which correspond to their resources overhead. Load units are assumed static and does not change in size throughout time. It is assumed that load transfers do not induce any temporary loads whether computation or communication. Depending on the approach, load can be either continuous or discrete.

4.1.3 Mathematical Model

Inspired by the model equation described in Eq. 2.1, the general equation that governs both of the presented models can be described as:

$$x_i^{(k+1)} = x_i^{(k)} + \sum_{j}^{n} f_{ij}\left(x_j^{(k)} - x_i^{(k)}\right) + \eta_i^{(k+1)} - \zeta_i^{(k)} \tag{4.1}$$

With n indicating the number of nodes present in the core network, $x_i^{(k)}$ representing the load at node i at time step k. The summation function f_{ij} is an abstract function that evaluates a signed quantity and is evaluated according to the direction of the load transfer such that

$$f_{ij}\left(x_j^{(k)} - x_i^{(k)}\right) = -f_{ji}\left(x_i^{(k)} - x_j^{(k)}\right) \tag{4.2}$$

ζ_i accounts for the calculations/executions that can be performed at each time step at node i thus discharging load. In contrast, the term η_i accounts for the newly appended tasks at node i.

Table 4.1: Models Comparison

Model	Overhead Model	Transfer Model	Network Parameters
Linear	• Load is numerically continuous	• Fixed load transfer ratio, centrally optimized once at initialization. • Load is transfered in synchronous clock cycles computed by a linear function. • Load transfer objective is to have unified load distribution	• Fixed network size • Unified node capacities
Non-linear	• Load is numerically discrete	• Load is transfered in synchronous clock cycles decentrally computed by a piecewise non linear transfer function • Load transfer objective is to have unified resource utilization	• Dynamic network size • Different node capacities

5 Methodology Investigation - Analytical Approach - Linear Model

5.1 Modeling of the Adaption Algorithms

In the multi core processing domain, load distribution constitutes a crucial factor to the performance and thus was investigated thoroughly. A model was introduced to describe mathematically the load transfer between the different cores in the discrete time domain in Sec. 2.7.2 where Eq. 2.1 is taken as the fundamental basis upon which the algorithm model is constructed.

5.2 Model Characteristics

In the linear model, the load units are considered to be numerically continuous. The load exchange between the nodes is governed by a linear transfer function that is executed in synchronous clock cycles. The linear function is constant (however can be optimized). The linear function parameters are considered to be time invariant. Furthermore, the network size and adjacency are considered to be static and does not change throughout time. The Optimization for the load exchange can be performed centrally which results in computing optimum load transfer parameters. The optimization can be performed with different criteria, e.g., minimizing the settling time, eliminating certain dynamics in the system response, etc.

5.3 Model

Using the load x as a state of the system and in discrete time domain (k), the system can be described as:

$$x_i^{(k+1)} = x_i^{(k)} + \sum_{j=1}^{n} f_{ij}\left(x_j^{(k)} - x_i^{(k)}\right) + \eta_i^{(k+1)} - \zeta_i^{(k)} \tag{5.1}$$

under the assumption of an undirected graph with a linear transfer function, the following equation holds:

$$x_i^{(k+1)} = x_i^{(k)} + \sum_{j=1}^{n} e_{ij} \cdot \alpha_{ij}\left(x_j^{(k)} - x_i^{(k)}\right) + \eta_i^{(k+1)} - \zeta_i^{(k)} \tag{5.2}$$

With n indicating the number of nodes present in the core network, $x_i^{(k)}$ representing the load at node i at time step k, transfer coefficient α_{ij} is a non negative constant that

evaluates to zero when both indices are identical. Instead of assuming that the transfer between non-connected nodes is zero, the model is adjusted to include the adjacency matrix such that e_{ij} is a switch constant the describes the adjacency between node i and j providing a switch control over the communication edges. The summation term is a vector quantity and can evaluate to a signed value according to the direction of the load transfer. ζ_i accounts for the calculations/executions that can be performed at each time step at node i thus discharging load. On the contrary, the term η_i accounts for the newly appended tasks at node i.

$$\psi_i = \eta_i - \zeta_i \tag{5.3}$$

Intuitively, a node cannot have an execution value that is higher than the corresponding available load making the node transaction ψ strictly state dependent. The bounding inequality can be represented as:

$$\zeta_i \leq \eta_i + x_i \tag{5.4}$$

ψ represents the inputs and outputs of the system.

In order to investigate the stability and the analysis of the different algorithms, an objective has to be defined. Since the objective of each algorithm depends on the executed scenario, the objective is set to the generic load balancing, i.e., the loads between the nodes are balanced so that each node has an equal load value $\hat{x}$.

$$\hat{x}^{(k)} = \frac{\sum_{i=1}^{n} x_i^{(k)}}{n} \tag{5.5}$$

The homogeneous problem can be considered by assuming that:

$$\psi_i = 0 \tag{5.6}$$

Furthermore, since the states stay unchanged at each time step and the system does not change the load profile, i.e., preserves the total load in the network, unless a distribution algorithm is executed. The system matrix A can then be described as

$$A = I_{n \times n} \tag{5.7}$$

excluding the algorithm overhead, it can be seen that the system is marginally stable, as the load amount in the system is only redistributed and the total amount is not affected.

The algorithm can be described using a feedback matrix K, and the resultant system can be thus described as:

$$\vec{x}[k+1] = A_K \vec{x}[k] \tag{5.8}$$

A mathematical description of Δx enables the computation of the system matrix A_K. The feedback matrix K can thus be interpreted from Δx,

$$\Delta \vec{x} = -K \cdot \vec{x} \tag{5.9}$$

$$K = \begin{bmatrix} \gamma_1 & -e_{12} \cdot \alpha_{12} & \dots & -e_{1n} \cdot \alpha_{1n} \\ -e_{21} \cdot \alpha_{21} & \gamma_2 & \dots & -e_{2n} \cdot \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -e_{n1} \cdot \alpha_{n1} & -e_{n2} \cdot \alpha_{n2} & \dots & \gamma_n \end{bmatrix} \tag{5.10}$$

with

$$\gamma_i = \sum_{j=1}^{n} e_{ij}\alpha_{ij} \tag{5.11}$$

The feedback controller system matrix A_K can be computed as

$$A_K = A - K = I - K \tag{5.12}$$

$$A_K = \begin{bmatrix} 1-\gamma_1 & e_{12}\cdot\alpha_{12} & \dots & e_{1n}\cdot\alpha_{1n} \\ e_{21}\cdot\alpha_{21} & 1-\gamma_2 & \dots & e_{2n}\cdot\alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{n1}\cdot\alpha_{n1} & e_{n2}\cdot\alpha_{n2} & \dots & 1-\gamma_n \end{bmatrix} \tag{5.13}$$

Since the network topology deduces the values of the e_{ij} elements, therefore analyzing the α_{ij} parameters can determine the stability boundary conditions and control the convergence and the system response dynamics.

5.3.1 Stability and Convergence Analysis

Due to the nature of the network symmetry and the nature of the load balancing problem, the A_K matrix is double stochastic and irreducible. According to Perron-Frobenius theorem [90] [49], all eigenvalues of A_K are bounded by the unit circle, i.e., stable system. However, stability does not indicate convergence. Computing the eigenvalues of matrix A will result in

$$-1 < \lambda_n \leq ... \leq \lambda_2 < \lambda_1 = 1 \tag{5.14}$$

With an eigenvalue at 1 (which can be intuitively interpreted as well due to the integrator behavior nature of the system).

$$\gamma_i = \sum_{j=1}^{n} \alpha_{ij} \tag{5.15}$$

As previously mentioned in Sec. 2.7.2, in case λ_n is equal to -1, the system does not converge. Furthermore, convergence can be ensured if one (or both) of the following holds:

- $1 - \gamma_i > 0$
- the network is not a bipartite graph

5.3.2 System Dynamic and Performance Analysis

In order to find the fastest settling time, Eq. 5.16 which determines the approximate settling time of the system within two percent can be used to derive the objective function shown in Eq. 5.17 [2] [8] [41] .

$$t_{s_{2\%}} \approx \ln(2\%) \cdot T \cdot \frac{1}{\ln\left(\max(|\lambda_2|, |\lambda_n|)\right)} \tag{5.16}$$

with T to be the discretization time.

$$f_0(\lambda) = -\frac{1}{\ln|\max(|\lambda_2|, |\lambda_n|)|} \tag{5.17}$$

Through pole placement, manipulation of the system settling time and limiting certain response dynamics like oscillations can be achieved. This can be extremely useful in the automation domain, since load oscillation is not desired in the network for the communication overhead it can result in. Minimizing the objective function f_0 reduces the settling time.

$$\min_{\lambda} f_0(\lambda) \tag{5.18}$$

The minimization should be performed with care in order to not violate the following boundary conditions and assumptions in order to ensure system stability and convergence $\forall i \in N$, Where $N = \{1, 2, .., n\}$, the following holds:

- At any given time, the node utilization cannot exceed the maximum resources capacity x_c

$$x_i^{(k)} \leq x_c \tag{5.19}$$

- All eigenvalues lie within the unit circle and are greater than -1

$$-1 < \lambda_i \leq 1 \tag{5.20}$$

- By definition α is a non negative constant (cf. Sec 5.1), to ensure that all elements of the A_K matrix are non negative and do not exceed 1, i.e., the transferred load ratio is non negative and does not exceed 1.

$$0 \leq \alpha_{ij} \leq 1 \tag{5.21}$$

- At each time step the net total node transactions must not exceed the existing load of the node.

$$\sum_{j=1}^{n} \alpha_{ij} \leq 1 \tag{5.22}$$

- A further constraint can be applied to Eq. (5.20) to constrict oscillations in the response.

$$0 \leq \lambda_i \leq 1 \tag{5.23}$$

  Important to realize, according to Abel-Ruffini theorem [4], the aforementioned minimization procedure is only analytically possible to perform in networks with maximum of five unknowns. Solving the characteristic equation is analytically impossible for equations with order higher than fourth degree, i.e., 5 poles result in 4 unknowns since the first pole is always 1 (since it is a discrete time domain and the system exhibits an integrator behavior), otherwise a numerical solution using brute force optimization is inevitable. Special graphs that contain symmetries, e.g., ring, star, Petersen, etc., can have a formula to calculate the optimal transfer factors α as shown in [15] or can be approximated using regression models as presented in Sec. 5.5.

5.4 Modeling of Multidimensional Loads

The aforementioned model describes the load transfer in one dimension. However, this is insufficient for modeling complex problems that have more than one load type involved. Performing optimization for different objectives simultaneously can cause conflicts between the optimization criteria. In this section, the unidimensional equations are extended to model a Multi-Dimensional (MD) load distribution system.

5.4.1 MD Problem Classification

The MD load distribution problem can be classified into the following:

Independent MD Problem

In the trivial case, where more than one dimension exists, however, independent of one another, the problem is simplified to many parallel uni-dimensional problems. In this case, each dimension is considered individually and the optimization is performed as already discussed.

Semi-Coupled MD Problem

In case of a semi-coupled system, the dimensions are dependent on one another. However, a restriction exists at the input, i.e., the manipulation of the load distribution can only be performed through certain dimensions and not all. Hence, an indirect manipulation will be needed if a manipulation of a load without an input is necessary.

Fully-Coupled MD Problem

In a fully coupled system, the dimensions are dependent on each other. Moreover, each dimension can be manipulated through its own input. A manipulation in one dimension will influence all other dimensions, however, not necessarily equally. The linearity or non-linearity of the influence should be defined according to the problem and designated optimization objective.

5.4.2 MD Problem Modeling

Utilizing the same equation system that was shown in the uni-directional load problem can act as a basis for the new MD system. Appending the following aspects to the problem description enables modeling MD problems:

- Cost function: In MD problems, not all loads have the same costs or weights. Thus, a cost/objective function is essential to describe the real weights of each dimension. This is realized in the evaluation model that will be introduced in the following section (cf. Sec. 5.4.2- Evaluation Model).
- Relativistic Transformation: As previously mentioned, manipulating a single load dimension can cause a non linear transformation in other dimensions. Accounting for these changes is essential and is realized in the relativistic transformation model 5.4.2- Relativistic Transformation Model).

Evaluation Model

The real overhead costs are demonstrated by the evaluation model.

$$Z_i = \sum_{d=1}^{D} G_d(x_{di}) \tag{5.24}$$

where G_d is the weight of load dimension d. In order to compare the different dimensions with their real weight, the loads have to be projected onto the dimension Z_i that represents the total weighted overhead of all dimensions of node i. For simplicity, a linear cost function is used:

$$Z_i = \sum_{d=1}^{D} G_d \cdot x_{di} \tag{5.25}$$

Relativistic Transformation Model

This model traces the disruptions or load manipulations caused by one dimension to other existing load dimensions. It should not be confused with the evaluation model, as this model does not represent the cost of a single unit rather it is solely responsible for relativistic load changes due to load transactions. For example, transferring a computation load between two nodes can cause a communication overhead between them. The relativistic transformation evaluates to:

$$\sum_{\substack{m=1 \\ m \neq d}}^{D} \sum_{j=1}^{n} T_{md} \cdot e_{ij} \cdot \alpha_{dij} \cdot \left| x_{mj}^{(k)} - x_{mi}^{(k)} \right| \tag{5.26}$$

For simplicity, T_{md} is assumed non-negative and constant. Contrary to semi-coupled systems, in a fully coupled system, an inverse transformation T_{dm} exists ideally in the form:

$$T_{dm} = T_{md}^{-1}$$

Where T_{dm} accounts for the reversed transformation (ideally is the reciprocal of T_{md}). Fig. 5.1 shows an example of the relativistic transformation of transfer occurring between two nodes.

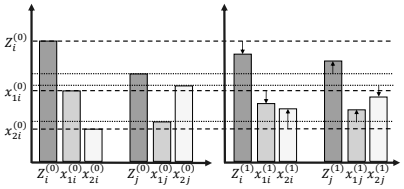


Figure 5.1: Illustration of the Relativistic Transformation Between Two Nodes

Load Transfer Model

The load transfer model acts as the coupler of all dimensions. The different dimensions are scaled with their real weights and projected onto one dimension Z where load transfers take place. This transforms the MD problem again into a uni-dimensional problem allowing the use of the conventional equations. Contrary to the uni-dimensional paradigm, where the average load stays constant throughout the load balancing process as shown in Eq. (5.5), the average load in the projection dimension varies with each load transfer and can be described as follows:

$$\hat{Z}^{(k)} = \frac{\sum_{i=1}^{n} Z_i^{(k)}}{n} \tag{5.27}$$

Rewriting Eq. 2.1 for the MD problem formulates to:

$$Z_i^{(k+1)} = Z_i^{(k)} + \sum_{j=1}^{n} \Gamma_{ij} \cdot \left(Z_j^{(k)} - Z_i^{(k)} \right) \tag{5.28}$$

Where the weighed transfer evaluates:

$$\Gamma_{ij} \cdot \left(Z_j^{(k)} - Z_i^{(k)} \right) = \sum_{d=1}^{D} G_d \cdot \left(e_{ij} \cdot \alpha_{dij} \left(x_{dj}^{(k)} - x_{di}^{(k)} \right) + \sum_{\substack{m=1 \\ m \neq d}}^{D} \sum_{j=1}^{n} T_{md} \cdot e_{ij} \cdot \alpha_{dij} \cdot \left| x_{mj}^{(k)} - x_{mi}^{(k)} \right| \right) \tag{5.29}$$

Substituting Eq. 5.29 in Eq. 5.28:

$$Z_i^{(k+1)} = Z_i^{(k)} + \sum_{j=1}^{n} \sum_{d=1}^{D} G_d \cdot \left(e_{ij} \cdot \alpha_{dij} \left(x_{dj}^{(k)} - x_{di}^{(k)} \right) + \sum_{\substack{m=1 \\ m \neq d}}^{D} \sum_{j=1}^{n} T_{md} \cdot e_{ij} \cdot \alpha_{dij} \cdot \left| x_{mj}^{(k)} - x_{mi}^{(k)} \right| \right) \tag{5.30}$$

Thus, the state space representation can be described by:

$$\vec{Z}[k+1] = A_{ZK} \cdot \vec{Z}[k] \tag{5.31}$$

Where A_{ZK} is the feedback controlled system matrix and can be computed as

$$A_{ZK} = A_Z - B_Z \cdot K = I - B_Z \cdot K_Z \tag{5.32}$$

Similar to the uni-dimensional problem, in the MD-problem, the system matrix A_Z is also interpreted as an identity matrix as the system preserves its previous state and does not cause any internal changes. The K matrix describes the load balancing algorithm feedback. The node state changes $\Delta\vec{Z}$ can be computed in the same manner as in uni-dimensional:

$$\Delta\vec{Z} = -K \cdot \vec{Z} \tag{5.33}$$

The K matrix can be interpreted as in Eq. (5.9), and hence matrix A_{ZK} can be described as

$$A_{ZK} = \begin{bmatrix} \gamma_1 & e_{12} \cdot \Gamma_{12} & \dots & e_{1n} \cdot \Gamma_{1n} \\ e_{21} \cdot \Gamma_{21} & \gamma_2 & \dots & e_{2n} \cdot \Gamma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{n1} \cdot \Gamma_{n1} & e_{n2} \cdot \Gamma_{n2} & \dots & \gamma_n \end{bmatrix} \tag{5.34}$$

with

$$\gamma_i = 1 - \sum_{j=1}^{n} e_{ij}\Gamma_{ij} \tag{5.35}$$

The elementary-dimensions can be represented as

$$\vec{x_d}[k+1] = A_{xK_d}[\vec{Z}] \cdot \vec{x_d}[k] + \sum_{\substack{m=1 \\ m \neq d}}^{D} B_{x_m} \cdot \Delta\vec{x_m}[k] \tag{5.36}$$

Where A_{xK_d} is the controlled feedback system matrix. The relativistic input matrix B_{x_m} which is responsible for the inputs of other states in case of a fully-coupled system (cf. Sec. 5.4.2- Relativistic Transformation Model). As previously mentioned, the load transfer computations are evaluated in the Z dimension which is not a realistic dimension. Thus mapping the multidimensional transfer factor Γ_{ij} to the unidimensional α_{dij} factors is essential to carry out the load transfer process. It can be clearly seen, that no unique solution for the controlled feedback system matrices A_{xK_d} can be derived.

Reformulating Eq. (5.29) so that α_{dij} is the subject of the formula requires a further relation which distributes the Γ_{ij} among the unidimensional α_{dij}. Otherwise, by default, the α_{dij} is tuned to receive equal proportions from the Γ_{ij} leading to convergence according to the adjusted cost function shown in Eq. (5.25). On the other hand, adjusting unequal proportions leads to a faster convergence in dimensions with higher proportions share. Another method to dynamically adjust the proportion is using game theory [65], thus establishing a Nash equilibrium according to the offered resources and loads. Alternatively the agent based combinatorial auction solution proposed in [21] or [38] can also be considered.

Realistic Model

The realistic model shows the real values of the load in their separate dimensions. This model reflects the real value of the load at each dimension at each time step. The model can be described with the following equation:

$$x_{di}^{(k+1)} = x_{di}^{(k)} + \sum_{j=1}^{n} e_{ij} \cdot \alpha_{dij} \left(x_{dj}^{(k)} - x_{di}^{(k)} \right) + \sum_{\substack{m=1 \\ m \neq d}}^{D} \sum_{j=1}^{n} T_{md} \cdot e_{ij} \cdot \alpha_{dij} \cdot \left| x_{mj}^{(k)} - x_{mi}^{(k)} \right| + \eta_{di}^{(k+1)} - \zeta_{di}^{(k)} \tag{5.37}$$

with the assumption that no external load addition or execution occur at the nodes, the equation can be simplified to the following:

$$x_{di}^{(k+1)} = x_{di}^{(k)} + \sum_{j=1}^{n} e_{ij} \cdot \alpha_{dij} \left(x_{dj}^{(k)} - x_{di}^{(k)} \right) + \sum_{\substack{m=1 \\ m \neq d}}^{D} \sum_{j=1}^{n} T_{md} \cdot e_{ij} \cdot \alpha_{dij} \cdot \left| x_{mj}^{(k)} - x_{mi}^{(k)} \right| \tag{5.38}$$

Simulation

As a proof of concept, the equations have been simulated in MATLAB, and a scenario with the following conditions was executed:

- A semi-coupled system
- 10 nodes that are connected in a ring topology
- Two dimensions were considered in which only one is reachable
- Parameters used: $T_{md} = 0.5$ and $G_d = 4$
- Balanced load in the reachable dimensions terminates the simulation

The start and final output of the simulation are shown in Fig. 5.2.

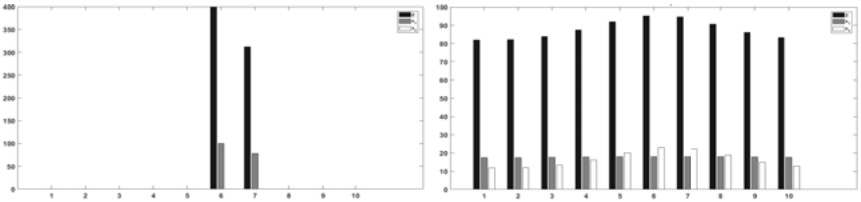


Figure 5.2: Semi-Coupled System - Start and Final Simulation Step of Combined Load Z, Load Dimensions x_1, and x_2 in black, gray (reachable dimension), and white resp.

5.5 Performance Enhancement via Regression Models

As previously shown, adjusting the performance of a distribution algorithm is not a simple task. Thus performing an online performance optimization can be time consuming and might even not be reasonable in case of topologies that are highly dynamic. In this section, an investigation is conducted to examine whether a regression model can compute the optimal values given the topology of a network to minimize the Markov chain mixing time. In this investigation, a hub spoke network topology is chosen for its ability to describe any topology in an abstract manner. However, as a starting point, a special topology of hub and spoke networks is considered. This investigation is yet to be extended to describe more variables which could possibly enable a description of a regression model that can compute the optimal values of the edge weights in linear time. Important to realize, the regression model does not compute the exact optimal edge weights rather approximate values that lie within an allowed error range from the industrial automation perspective and in return the model provides a significant improvement in the computation complexity thus reducing processing time (cf. Sec. 5.5.3).

5.5.1 Optimizing the Transfer Coefficient

In the presented optimization problem, the objective is to compute the fastest mixing time of the Markov chain induced by the matrix presented in Eq. 2.4. Due to the complexity of such an optimization, reducing the number of free parameters can simplify the problem. Given a symmetry in a network, [14] shows that with a fixed-point of the automorphism group of the underlying graph, an optimum solution comprising the same symmetry exists. In other words, symmetrical edges can be considered equal.

5.5.2 Ring Hub and Spoke Networks

In the shown regression, the Hub and Spoke (HS) topology is classified into two major classes, a homogeneous and heterogeneous HS topologies. The former describes a perfect HS network indicating identical hubs (also denoted as 'h') , i.e., each hub is connected to the same number of independent spokes (also denoted as 's'), among the hubs a ring topology is formed as each hubs is connected to two neighboring hubs. This topology is chosen for its high symmetry. The edge weights between a hub and another hub and between a hub and a spoke will be henceforth denoted as α and β, respectively, as shown in Fig. 5.3 where 3 hubs and 4 spokes are illustrated. On the other hand, heterogeneous HS networks has less symmetry and allows irregular architectures to be included, e.g., different number of spokes per hub, different number of neighbors for each hub, etc. HS topology has been chosen as a basis for the regression model. It can be used as an abstract topology to derive any other existing topology and build complexer models in the future. The ring topology for instance is one special form of HS where the network consists only of hubs. As a result, the HS topology is chosen to be a start point for a regression model which can be later adjusted to fit other topologies.

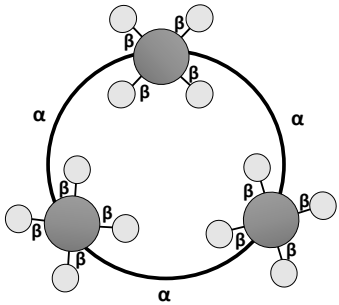


Figure 5.3: A Perfect Hub-Spoke Network $h = 3$ and $s = 4$

Using the induced Markov chain shown in Eq. 2.4 and with the following definition of an equilibrium distribution

$$\lim_{t\to\infty} \vec{x}^{(t)} = \hat{x} \cdot \mathbb{1}, \quad \hat{x} = \frac{1}{n}\sum_{i=1}^{n} x_i^{(t)}. \tag{5.39}$$

As shown in [13] and [56], the minimization of the SLEM can be formulated as a SemiDefinite Program (SDP):

$$\begin{aligned} \text{minimize} \quad & \beta \\ \text{subject to} \quad & -\beta I \preceq A - \frac{1}{n}\mathbb{1}\mathbb{1}^T \preceq \beta I, \\ & A_{ij} \geq 0,\ A = A^T,\ A\mathbb{1} = \mathbb{1}. \end{aligned} \tag{5.40}$$

Initially, a SDP for the input parameters $h,\ \beta$ has to be constructed.
Let, without loss of generality, the first h nodes represent the hubs inside the network.

Assuming the symmetry of coefficients and considering the aforementioned notation, the transition matrix $A(h,s)$ has the following form:

$$\begin{aligned}
&A_{i,i+1} = A_{i+1,i} = \alpha,\ 1 \le i \le h-1;\\
&A_{1,h} = A_{h,1} = \alpha;\\
&A_{ij} = A_{ji} = \beta,\ (i-1)s + h + 1 \le j \le is + h,\ 1 \le i \le h;\\
&A_{ii} = 1 - 2\alpha - s\beta,\ 1 \le i \le h;\\
&A_{ii} = 1 - \beta,\ h+1 \le i \le n;\\
&A_{ij} = 0,\ \text{otherwise.}
\end{aligned}$$

In this scheme, within a defined network, i.e., s and h are known, α, β are variables. Important to realize, the A matrix diagonal elements, i.e., A_{ii}, are bounded by the A matrix stochasticity, i.e., $A\mathbb{1} = \mathbb{1}$. Thus the objectives and constraints for the SDP can be formulated and inputted to the solver. Denoting $\mathcal{E} = \frac{1}{n}\mathbb{1}\mathbb{1}^T$ and introducing the parameter γ:

$$\begin{aligned}
&\text{minimize } \gamma\\
&\quad \text{s.t. } \{\forall i,j\ A_{ij} \ge 0,\ A\mathbb{1} = \mathbb{1},\ A + \gamma I - \mathcal{E} \succeq 0,\\
&\qquad -A + \gamma I + \mathcal{E} \succeq 0,\ \gamma \ge 0\}
\end{aligned}$$

Computing parameters for small networks

Using the SDPT3-solver for the constructed SDP, computing solutions $(P, \gamma(P))$ for networks with $3 \le h \le 9$ and $2 \le s \le 19$ with high precision (the dual gap set to 10^{-10}) is possible. The optimal solutions for the coefficients α and β can be determined from the distribution matrices A:

$$\alpha = A_{1,2},\ \beta = A_{1,h+1}$$

Predicting the coefficients

Using the obtained data for small networks, a regression analysis is conducted to investigate the relationship between the transition coefficient α and β as function of the network parameters h, s. To establish a model, the following approach is utilized: the sum of the edge coefficients incident to a hub must be equal to 1, such that the corresponding diagonal elements of A evaluate to 0. This assumption is inspired by [22]: such transition matrices A are optimal in the set of linear modifications $\{(A+kI_n)/(1+k) \mid k \ge \min_j a_{jj})\}$. Setting $a_{ii} = 0$ for $i = 1 \dots h$ yields the dependence $\alpha = \frac{1-s\beta}{2}$ (for $h > 2$). Using the data analysis, a guess for modeling β can be made. Overall, the model constructed for regression is

$$\frac{1}{\beta} = a \cdot s + b \cdot h \cdot s + c \cdot h + d, \quad \alpha = \frac{1 - s\beta}{deg(h) - s}, \tag{5.41}$$

where $deg(h)$ denotes a hub's subset in its valency. a, b, c and d denote the coefficients to be evaluated.

5.5.3 Regression Model

Special Cases

The special case $h = 2$ (and hence $deg(h) = s + 1$) is considered separately, since as aforementioned, in the analysis, only undirected graphs are considered, which in the common sense, a ring topology cannot be constructed by two hubs. The regression model shown in Eq. (5.41) is applied to a dataset with $s = 1 \dots 20$. Fig. 5.4 demonstrated the work flow of the used approach.

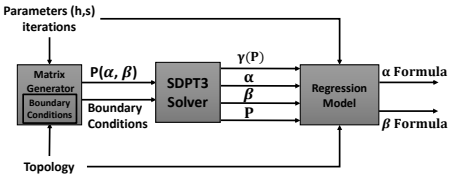


Figure 5.4: Approach Work Flow

The case $h = 1,\ s \geq 2$ depicts a star network topology with a single unknown parameter β (since all edges are symmetric) in the transition matrix. The optimization problem can thus be solved analytically. Eigenvalues of $A(\beta)$ are $1,\ 1 - \beta,\ 1 - (s + 1)\beta$ (with multiplicities $1,\ s - 1,\ 1$ respectively). Minimizing $\gamma(P)$ under the constraint $s \cdot \beta \leq 1$ yields $\beta = \frac{1}{s}$.

Ring HS topology

The described regression model yields the following expression for computing the transition coefficients:

$$\begin{aligned} \beta^{-1} = &\quad (1.1673 + 0.2097 \cdot h)s + (0.1705 \cdot h - 0.1769) \\ \alpha = &\quad \frac{1}{2}(1 - s \cdot \beta) \end{aligned} \tag{5.42}$$

Further instances of HS (including networks of larger scale, e.g., $h \geq 100$) were evaluated using the obtained model. The outputs of the regression model and the solver are compared using the settling time (cf. Eq. (5.16)) as a performance criterion as shown in the validation work flow in Fig. 5.5.

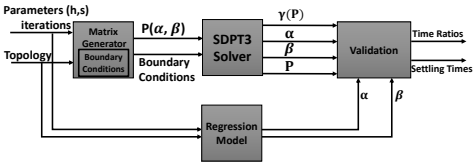


Figure 5.5: Validation Work Flow

A sample of the validation results is demonstrated in Table 5.1. The table is divided into four sections. The first sections shows two columns denoted with 'h' and 's' indicate the number of hubs and spokes of the analyzed network respectively. The second and third sections of the table show the results of the solver and the regression model respectively. The two sections comprise four columns showing the α, β (cf. Fig. 5.3), $\gamma(P)$ (cf. Sec. 5.5.2) and the settling time of the network using Eq. (5.16) respectively. The final column shows the time ratio which evaluates the settling time of the solver to the regression ratio. The settling times in small networks evaluated by the regression model show high precision with an error within the 0.05% tolerance interval. On the other hand, in bigger networks, e.g. $h = 100, 200$, the regression model outperforms the solution provided by the solver (results shown in bold font in the table). This is due to the operation time out or numerical problems which leads to producing a non-guaranteed optimum solution. The results are plotted using a surf for better illustration in Fig. 5.6 with the settling time, the number of hubs and the number of spokes on the axes. The surf demonstrates the regression modeled data while the diamond data set demonstrates the solver results. At the data point with 100 hubs and 20 spokes, it can be seen that the regression model outperforms the solver computation.

A note on complexity

In general, SDP falls under the class of conic programs in the classification of convex problem optimization, which is recognized as efficiently approximable in polynomial time in most cases. In particular, polynomial methods are applicable for the minimization problems as formulated in Eq. (2.10) [66]. In the computation stage, the solver failed to optimize several middle-sized instances due to program terminations. The terminations are caused by lack of progress in infeasibility (termination code = -9) or numerical problems such as deterioration of dual infeasibility (SDPT3 termination code = -7). Moreover, in larger network instances, runtime becomes also an issue. Generally, the best proven bound for the number of iterations for infeasible interior-point algorithms (which is used by the utilized solver) is $\mathcal{O}(\sqrt{n}L)$, where L is the bit-length of input [71]. Each computation iteration executes a matrix Cholesky decomposition, which requires up to $\mathcal{O}(n^3)$ steps. Furthermore, the high symmetry of the semidefinite matrix does not reduce significantly the CPU-time [53], as well as, after reducing the problem to two parameters using the symmetry, the problem remains difficult to compute. The complexity of the problem is not simple to estimate, however, minimizing the polynomial roots can be considered as the lower bound for the complexity since it is an essential step for the eigenvalue optimization problem. Moreover, according to Abel-Ruffini theorem [4] [6], for $n \geq 5$, the problem becomes analytically intractable. Therefore, utilizing iterative numerical methods is inevitable. The aim of the regression model is to construct a near-optimal transition matrices for large networks which can compute the problem in $\mathcal{O}(1)$ runtime which is applicable for highly dynamic networks. With such a trade-off, sacrificing a negligible accuracy (which in reality has limited consequences, since loads are discrete and not continuous) for a guaranteed low execution time can provide an alternative to supersede the semidefinite optimization method.

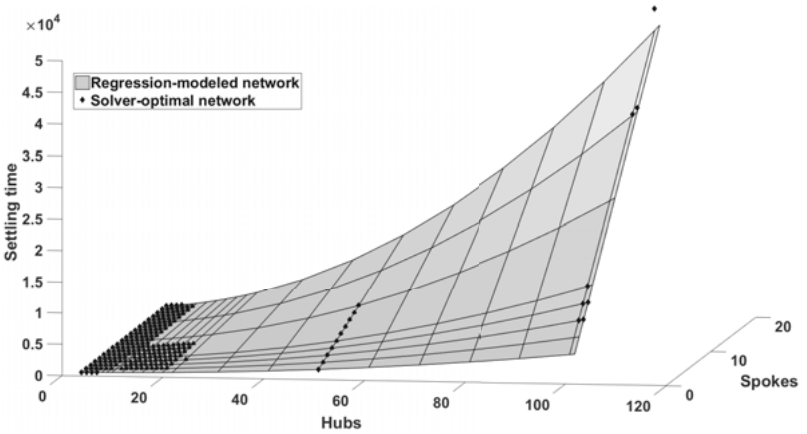


Figure 5.6: Solver and Regression Model Results Surf

Table 5.1: Settling time comparison

		Optimization with SDPT3				Regression				
h	s	α	β	$\gamma(A)$	Settling time, s	α	β	$\gamma(A)$	Settling time, s	Time ratio
3	2	0.2500	0.2500	0.862372	26.420441	0.2454	0.2546	0.862425	26.431269	1.000410
3	4	0.2380	0.1310	0.927878	52.260983	0.2340	0.1330	0.927896	52.275243	1.000273
3	7	0.2325	0.0764	0.957920	90.996405	0.2289	0.0775	0.957929	91.016348	1.000219
4	2	0.2778	0.2222	0.888889	33.213807	0.2786	0.2214	0.888890	33.214267	1.000014
4	4	0.2647	0.1176	0.941176	64.528616	0.2655	0.1172	0.941177	64.529366	1.000012
4	7	0.2586	0.0690	0.965517	111.481289	0.2594	0.0687	0.965518	111.482399	1.000010
5	2	0.3025	0.1975	0.910343	41.646700	0.3042	0.1958	0.910348	41.649081	1.000057
5	4	0.2888	0.1056	0.952062	79.633084	0.2903	0.1048	0.952064	79.636494	1.000043
5	7	0.2823	0.0622	0.971767	136.597473	0.2838	0.0618	0.971768	136.602423	1.000036
7	2	0.3405	0.1595	0.939345	62.520220	0.3409	0.1591	0.939346	62.520479	1.000004
7	4	0.3265	0.0867	0.967016	116.636094	0.3270	0.0865	0.967016	116.636498	1.000003
7	7	0.3198	0.0515	0.980417	197.800221	0.3202	0.0514	0.980417	197.800816	1.000003
10	2	0.3776	0.1224	0.962772	103.113200	0.3759	0.1241	0.962774	103.120292	1.000069
10	4	0.3644	0.0678	0.979376	187.716736	0.3629	0.0686	0.979377	187.725574	1.000047
10	7	0.3578	0.0406	0.987643	314.617478	0.3564	0.0410	0.987643	314.628055	1.000034
19	2	0.4290	0.0710	0.986587	289.690226	0.4252	0.0748	0.986593	289.828376	1.000477
19	4	0.4189	0.0405	0.992346	509.142154	0.4155	0.0422	0.992349	509.318403	1.000346
19	7	0.4137	0.0247	0.995344	838.318253	0.4105	0.0256	0.995346	838.551421	1.000278
50	2	0.4713	0.0287	0.997661	1670.648535	0.4684	0.0316	0.997662	1671.415036	1.000459
50	4	0.4663	0.0168	0.998627	2846.968597	0.4636	0.0182	0.998627	2847.972118	1.000352
50	**7**	**0.4636**	**0.0104**	**0.999153**	**4615.429603**	**0.4611**	**0.0111**	**0.999152**	**4612.785908**	**0.999427**
100	4	0.4827	0.0087	0.999632	10630.821615	0.4810	0.0095	0.999632	10634.025654	1.000301
100	5	0.4820	0.0072	0.999694	12793.433327	0.4804	0.0078	0.999694	12796.922991	1.000273
100	**15**	**0.4797**	**0.0027**	**0.999887**	**34621.892216**	**0.4785**	**0.0029**	**0.999886**	**34425.881960**	**0.994339**
100	**20**	**0.4342**	**0.0065**	**0.999920**	**48723.283682**	**0.4782**	**0.0022**	**0.999914**	**45240.371150**	**0.928516**
101	4	0.4829	0.0086	0.999639	10837.673686	0.4812	0.0094	0.999639	10840.285940	1.000241
101	**15**	**0.4673**	**0.0043**	**0.999890**	**35655.753526**	**0.4787**	**0.0028**	**0.999889**	**35089.745649**	**0.984126**
200	**4**	**0.4899**	**0.0049**	**0.999905**	**41118.560542**	**0.4903**	**0.0048**	**0.999905**	**41070.327441**	**0.998827**
200	**5**	**0.4830**	**0.0065**	**0.999922**	**49855.238805**	**0.4900**	**0.0040**	**0.999921**	**49355.314036**	**0.989972**

6 Methodology Investigation - Empirical Approach - Non-Linear Model

In the previous chapter, a mathematical analysis of load distribution is presented that can ensure the stability as well as enhance the performance. However, this analysis lacks many realistic aspects of the available algorithms for different reasons, e.g., the load is considered continuous and not discrete.

6.1 Model Characteristics

In the non-linear model, the load units are considered to be numerically discrete. The load exchange between the nodes is governed by a non linear piecewise transfer function that is executed in synchronous clock cycles. The piecewise function varies according to the algorithm's logic. Thus the transfer function and the load transfer coefficients are time variant. The optimization is performed decentrally since each node is able to retrieve information via its neighbors. Furthermore, the network size and adjacency are considered to be dynamic and can change at any point of time. In a decentral behavior, the optimization is hardly affected by changes in the global model (e.g., network size) as each node considers only a local model.

6.2 Performance Assessment

In order to assess the performance of algorithms, a list of indicators is needed to compare the performances in different conditions. Various Key Performance Indicators (KPIs) have been defined in [28], [78], [27], [77], [54], [54], [72], [37], [63], [80] and [46]. However, these KPIs were either not adequate or not precise enough for the presented work. A benchmark is presented where a list of KPIs are defined to measure the performance of distribution algorithms in [30]. In this chapter, the benchmark and the KPIs list are discussed. The distribution algorithms view the load distribution from an empirical perspective (cf. Table 4.1). Although the previously set objective in Eq. 5.5 is further utilized in this section, since discrete load packets are considered, it is important to realize that the objective value might not be reached in case the computed mean is not a whole number.

6.3 KPIs Preliminaries

In order to define a clear list of KPIs, a list of preliminaries that set the framework in which the KPIs are computed has to be defined to ensure fairness and consider the different perspective of the performance:

6.3.1 Network Topology

Different topologies can influence the performance of a certain algorithm. Thus it is essential to evaluate an algorithm in the different topologies, i.e., an algorithm can perform well in a “Star” topology but not necessarily in a “Mesh” topology.

6.3.2 Load Description

As previously mentioned, the load is considered in discrete load packets. A packet symbolizes the least measurable and transferable unit load.

6.3.3 Initial Conditions

Initial conditions, e.g., the starting load distribution profile, play a crucial role in the evaluation process. Two factors can influence the initial load distribution which are:

- The percentage of nodes occupied by the initial load, e.g., spike test can be performed by spawning a spontaneous load at a node and evaluate how the algorithm reacts
- The connectedness of the nodes carrying the initial load, i.e., how well connected a node that is carrying the initial load can affect the performance, e.g., in a star topology adding a spike load at the hub does not impose the same conditions as adding it at a spoke.

Other initial factors might exist and have to be defined according to the use case.

6.3.4 Node Capacity

It is assumed that all nodes have the same resource capacities. Alternatively, if this is not the case, the utilization percentage should be used as a measure which would normalize the different capacities. Point often overlooked is the objective in Eq. 5.5 which must be changed to an equal utilization of all nodes to fit the problem description.

6.3.5 Probabilistic Algorithms

A crucial point that must be taken into consideration is whether the algorithm is deterministic or probabilistic. In case of probabilistic algorithms, the KPI list must be evaluated an adequate number of times to account for variances and establish a valid (defined) confidence interval.

6.3.6 Foreknowledge of Terminating Conditions

The evaluation of an algorithm requires the foreknowledge of the termination condition in order to detect when the optimal (or in some cases final) distribution is reached. Since the considered load is discrete, the optimal distribution is not necessarily a whole number thus in case of an optimal distribution and the optimal distribution is said to be achieved when the loads of the nodes lie between the upper or lower bounds of the defined terminating value (cf. Eq. 5.5).

6.4 KPIs

The list has two types qualitative and quantitative KPIs.

6.4.1 Qualitative KPIs

The qualitative list comprises four KPIs:

Determinism

A boolean KPI that indicates whether the algorithm is deterministic or probabilistic

Initiation

This KPI indicates one of three possibilities: “sender”, “receiver” or “both”. This KPI gives an insight into how the load transfer process is triggered.

Stability

The stability KPI indicates whether the algorithm is “unstable”, “marginally stable” or “asymptotically stable”. The value given is based on the analysis shown in the quantitative stability KPI (cf. Sec 5.3.2).

Repeatability

On the one hand, in deterministic algorithms, the KPI describes the utilized strategy/logic to obtain repeatable results. On the other hand, in probabilistic algorithms, the KPI describes the utilized randomness logic which can be used to construct the probabilistic model of the algorithm.

6.4.2 Quantitative KPIs

Execution Time (T_e)

This KPI measures the average elapsed time till an algorithm converges to the final (optimal) load profile. The measured time here provides a stable and a reliable value as the measured quantity should specifically measure the CPU time consumed.

$$T_e = CPU_{time}^{(t_f)} - CPU_{time}^{(t_0)} \tag{6.1}$$

Where t_0 and t_f are the start and final times of the simulation respectively.

Time Steps (T_s)

The benchmark executes the algorithm in a discrete time domain. The T_s KPI provides an overview of the average elapsed iteration time steps until the final load profile is reached.

Total Square Error (TSE)

The TSE KPI measures the sum of the squared error of the loads at each node measured at the final distribution and the load distribution of each iteration time step of the algorithm execution. The KPI serves as a measurement of how close the distribution is at each step from the final profile.

$$TSE^{(t)} = \left[\sum_{i=1}^{n} \left(w_i^{(t)} - \lfloor \hat{w} \rfloor\right)^2\right] - c_0 \tag{6.2}$$

$$c_0 = n\left(\hat{w} - \lfloor \hat{w} \rfloor\right) \tag{6.3}$$

Where:
$\lfloor x \rfloor$ indicates a floor function that approximates the enclosed argument x (if not integer) to the prior greatest integer.

Stability

The stability of an algorithm can be measured using the TSE KPI. Proving that the state progression of the nodes does not reach a worse TSE state indicates asymptotic stability. This can be shown in the proof below by considering a single load transfer c between two nodes X and Y that have loads w_j and w_i respectively :

$$TSE^{(t+1)} \leq TSE^{(t)} \tag{6.4}$$

$$c = X(t+1) - X(t) = Y(t) - Y(t+1) \tag{6.5}$$

$$X(t) = w_i^{(t)} - \hat{w} \quad \text{and} \quad Y(t) = w_j^{(t)} - \hat{w} \tag{6.6}$$

$$TSE^{(t+1)} - TSE^{(t)} = X(t+1)^2 + Y(t+1)^2 - X(t)^2 - Y(t)^2 \tag{6.7}$$

$$X(t+1)^2 + Y(t+1)^2 \leq X(t)^2 + Y(t)^2 \tag{6.8}$$

$$(X(t) + c)^2 + (Y(t) - c)^2 \leq X(t)^2 + Y(t)^2 \tag{6.9}$$

$$2c^2 + 2 \cdot c\left(X(t) - Y(t)\right) \leq 0 \tag{6.10}$$

since $c > 0$, dividing both sides by $2c$

$$0 < c \leq Y(t) - X(t) \tag{6.11}$$

$$0 < c^{(t)} \leq w_j^{(t)} - w_i^{(t)} \tag{6.12}$$

Eq. 6.12 establishes that for each transfer of c units, the stability KPI is evaluated as either a marginal or asymptotic stable depending on the right inequality, if an equal sign is included or not respectively.

Scalability and Convergence Progression (SCP)

Another pivot point, that was lacking in the literature, is a KPI that determines the scalability of an algorithm as well as the performance of the convergence progression. The SCP KPI is a matrix that consists of arrays. Each array represents different network sizes n (in exponents of 10) demonstrating the TSE progression over different time progression points t_i. t_i is computed using the T_s and the progression constant K; the reciprocal of the length of the desired progression, e.g., if the desired progression points should show the quarters of the full progression, then K should be assumed as 4.

$$t_i = \frac{T_s \cdot i}{k} \text{ , where i=1...K} \tag{6.13}$$

$$SCP = TSE_{n_j}^{(t_i)} \tag{6.14}$$

$$SCP = \begin{bmatrix} TSE_{10}^{(t_1)} & TSE_{100}^{(t_1)} & TSE_{1000}^{(t_1)} & \dots & TSE_{n}^{(t_1)} \\ TSE_{10}^{(t_2)} & TSE_{100}^{(t_2)} & TSE_{1000}^{(t_2)} & \dots & TSE_{n}^{(t_2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ TSE_{10}^{(t_k)} & TSE_{100}^{(t_k)} & TSE_{1000}^{(t_k)} & \dots & TSE_{n}^{(t_k)} \end{bmatrix} \tag{6.15}$$

In order to have a more compact KPI as having a matrix is not optimal for evaluation purposes, a derived KPI Differential SCP ($DSCP$) is computed. The $DSCP$ uses the gradients of the TSE and applies decreasing weights for the gradients to give an edge to algorithms that converges faster in the initial time steps. The KPI is then normalized using the square of n and the k factor.

$$DSCP_n = \frac{1}{k \cdot n^2} \cdot \sum_{i=1}^{k} TSE'^{(t_i)}_{n} \cdot (k - i + 1) \tag{6.16}$$

$$TSE'^{(t_i)}_{n} = \frac{\partial TSE_n}{\partial t} \approx \frac{TSE_n^{(t_{i+1})} - TSE_n^{(t_i)}}{t_{i+1} - t_i} \tag{6.17}$$

Overhead Expenditures (OE)

A pivot point often overlooked is the resources consumption of the algorithm itself. The Overhead Expenditures (OE) aims to measure the computation, communication and memory overhead introduced by the algorithm using three sub KPIs.

- Memory Overhead Expenditures (MOE): Memory can be measured using two indicators, MOE_{Max} and $\overline{MOE}$ representing the maximum used memory and the overall average memory overhead per node throughout the execution respectively.

$$MOE_{Max} = \frac{M_{Max}}{n} = \frac{Max(M_{t_i..T_s}) - M_{t_0}}{n} \tag{6.18}$$

$$\overline{MOE} = \frac{\sum_{i=1}^{T_s}(M_{t_i} - M_{t_0})}{n \cdot T_s} \tag{6.19}$$

  With M_{Max}, M_{t_i}, M_{t_0} indicating the maximum memory consumption reached, consumed memory at time step i, and initial memory consumption used by computation node before the algorithm is executed respectively.

- Communication Overhead Expenditures ($CmOE$): The communication overhead requires to consider the overhead induced by the load transferred between nodes $CmOE_{Load}$ as well as the requests needed to initiate such a load transfer $CmOE_{MRQ}$. The communication delay can be computed as follows:

$$\beta = \beta_{pc} + \beta_q + \beta_t + \beta_{pg} \tag{6.20}$$

where β_{pc}, β_q, β_t, and β_{pg} are the routers processing, routing queues, transmission, and propagation delays respectively. Moreover, the load oscillation must be accounted for by inducing a punishment cost function $C(w_{ijk})$ each time a load transfer is initiated. The punishment cost is a function of the load transferred. One possible cost function can be equal to the product of load value and the number of hops of each transfer.

$$CmOE_{Load} = \sum_{i=1}^{T_s} \sum_{j=1}^{n} \sum_{k=1}^{m} C(w_{ijk}) \cdot \beta_k \tag{6.21}$$

where m is the number of load transfers executed per node and β_k is the corresponding communication latency for its route at iteration step i. The three summation terms consider the route communication latency for each message transmission route, by each sender node, at each time step taken during the algorithm execution respectively.

$$CmOE_{MRQ} = \sum_{i=1}^{T_s} \sum_{j=1}^{n} \sum_{k=1}^{m} Length(MRQ_{ijk}) \cdot \beta_k \tag{6.22}$$

$$CmOE_{total} = \frac{CmOE_{MRQ} + CmOE_{Load}}{n \cdot T_s} \tag{6.23}$$

- Computation Overhead Expenditures ($CpOE$): The computation overhead can be easily computed by the following formula

$$CpOE = \frac{T_e}{T_s} \tag{6.24}$$

6.4.3 Modular Benchmark

The benchmark offers a modular architecture such that different algorithms can be tested and the conditions can be tailored as desired. The benchmark generates a node map using the user input, e.g., network topology, initial load percentage, etc. Moreover it allows the user to provide an initialization function to reset and adjust the different parameters of the algorithm inputted at each iteration. The benchmark evaluates the aforementioned KPI list and iterates the execution to obtain a defined confidence interval as well as to measure the scalability of the algorithm by increasing the number of nodes. The flowchart of the benchmark is shown in Fig. 6.1.

6.4.4 Benchmark Testing

Testing Parameters

Different algorithms from the literature (cf. Sec. 2.7.1), were tested in star and full mesh network topologies. The benchmark was applied to two algorithms. The Algorithms

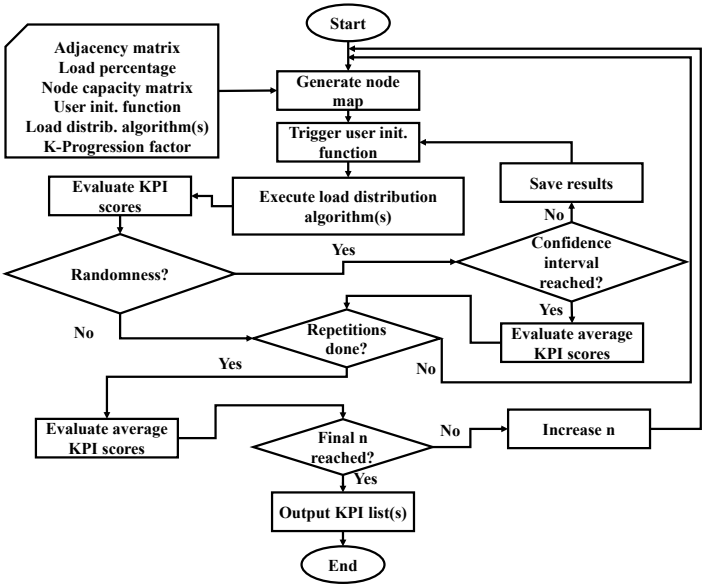


Figure 6.1: Benchmark Control Flowchart

implements the concepts presented in [16] and [47].The experiments are repeated for sizes of 100 nodes and 1000 nodes networks. In order to ensure fairness within the random parameters (e.g., initial load locations), the experiment used 15 different network map generations for each node count and each network map is repeated 10 iterations, i.e., a total of 300 iterations for each node counts. The initial load is randomly assigned to 20% of the nodes at each iteration. It is assumed that all nodes are homogeneous, i.e., they have the same capacity of 100 load points. The results are demonstrated with a 95% confidence interval. The k factor is assigned a value of 4, i.e., 25% progression intervals are measured.

Results

Table 6.3 shows the qualitative KPIs of both algorithms. The results of the star and full mesh topologies are shown in Tables 6.1 and 6.2 respectively. The prevailing performance is highlighted in bold font. Fig. 6.2 and 6.3 show the 100 and 1000 nodes TSE respectively. The final points of the TSE values are intentionally placed at 1 instead of 0 for illustration on logarithmic scales.

Table 6.1: KPI List for Star Topology

KPI	100 Nodes Mean + 95% CI	1000 Nodes Mean + 95% CI
Fluid T_e	**1.51 [1.26, 1.75]**	**136.67 [107.66, 165.67]**
Ants T_e	9.68 [9.08, 10.27]	566.51 [549.62, 583.42]
Fluid T_s	**308.09 [250.42, 365.75]**	4282.4 [3363.1, 5201.8]
Ants T_s	381.80 [358.44, 405.16]	**1338.2 [1303.8, 1372.7]**
Fluid $CpOE$	**0.0050 [0.0048, 0.0051]**	**0.0320 [0.0314, 0.0326]**
Ants $CpOE$	0.0254 [0.0250, 0.0258]	0.4234 [0.4168, 0.4299]
Fluid SOE_{max}	**0.0000 [0.0000, 0.0000]**	**0.0000 [0.0000, 0.0000]**
Ants SOE_{max}	0.0029 [0.0028, 0.0030]	0.0032 [0.0032, 0.0032]
Fluid $\overline{SOE}$	**0.0000 [0.0000, 0.0000]**	**0.0000 [0.0000, 0.0000]**
Ants $\overline{SOE}$	0.0020 [0.0020, 0.0021]	0.0023 [0.0023, 0.0023]
Fluid $CmOE$	**0.2143 [0.1826, 0.2460]**	**0.0184 [0.0153, 0.0216]**
Ants $CmOE$	0.7986 [0.7923, 0.8049]	0.7144 [0.7112, 0.7176]
Fluid $DSCP$	**-23.40 [-25.69,-21.76]**	**-0.025 [-0.031, -0.021]**
Ants $DSCP$	-3.60 [-3.52, -3.68]	-0.006 [-0.006, -0.006]

Table 6.2: KPI List for Mesh Topology

KPI	100 Nodes Mean + 95% CI	1000 Nodes Mean + 95% CI
Fluid T_e	**1.25 [0.73, 1.77]**	235.48 [84.74, 386.23]
Ants T_e	2.88 [2.73, 3.03]	**36.51 [33.94, 33.09]**
Fluid T_s	**14.51 [8.52, 20.51]**	**30.94 [11.32, 50.56]**
Ants T_s	128.15 [121.58, 134.72]	149.9 [137.6, 162.2]
Fluid $CpOE$	0.0860 [0.0841, 0.0879]	7.5294 [7.4036, 7.6552]
Ants $CpOE$	**0.0225 [0.0220, 0.0230]**	**0.2443 [0.2384, 0.2502]**
Fluid SOE_{max}	**0.0000 [0.0000, 0.0000]**	**0.0000 [0.0000, 0.0000]**
Ants SOE_{max}	0.0020 [0.0019, 0.0020]	0.0021 [0.0020, 0.0022]
Fluid $\overline{SOE}$	**0.0000 [0.0000, 0.0000]**	**0.0000 [0.0000, 0.0000]**
Ants $\overline{SOE}$	0.0015 [0.0015, 0.0016]	0.0016 [0.0016, 0.0017]
Fluid $CmOE$	4.5221 [3.7284, 5.3158]	3.1433 [2.4948, 3.7919]
Ants $CmOE$	**1.0719 [1.0463, 1.0974]**	**0.9991 [0.9738, 1.0244]**
Fluid $DSCP$	**-27.36 [-44.65,-20.22]**	-0.014 [-0.039, -0.009]
Ants $DSCP$	-11.57 [-11.61, -11.52]	**-0.090 [-0.096, -0.085]**

Table 6.3: Qualitative KPIs

KPI	Fluid	Ants
Determinism	No	No
Initiation	Sender	Sender
Model Stability	Marginal	Asymptotic
Repeatability	Round-Robin	Round-Robin

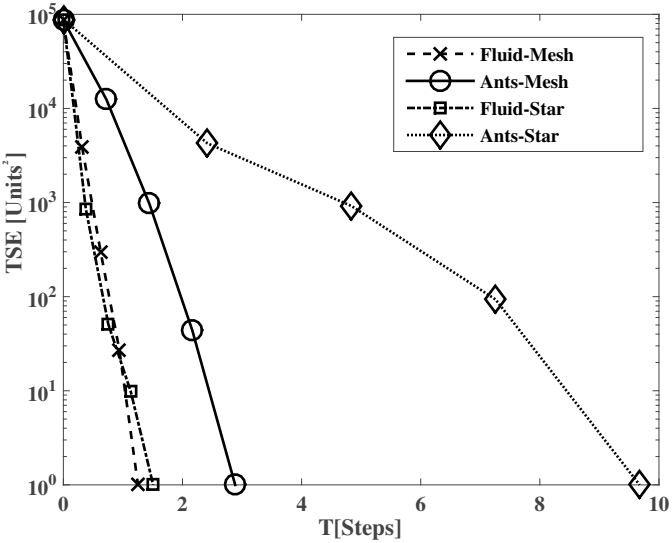


Figure 6.2: TSE Progression of 100 Nodes Network

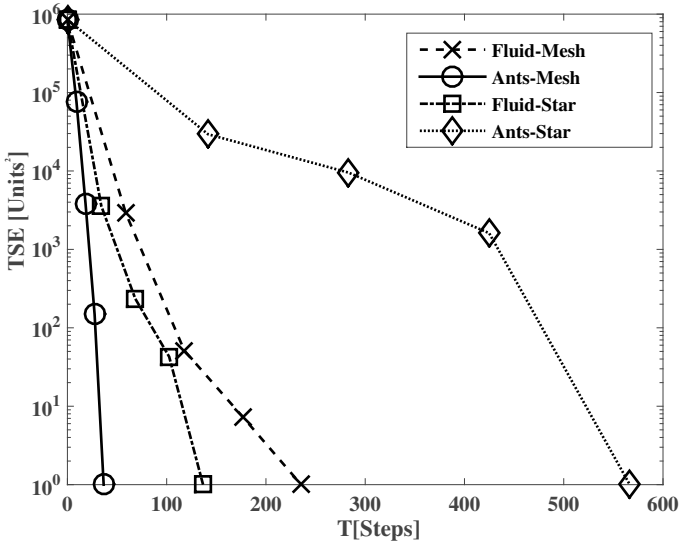


Figure 6.3: TSE Progression of 1000 Nodes Network

7 Use-Case - Implementation Approach

In the previous two chapters, an analytical and an empirical approach are presented. These approaches provide an insight into the system dynamics and the different algorithm implementations in the different domains. However, these approaches cannot be implemented in the automation domain due to a considerable reality gap. The reality gap can be seen mainly in the load model. In order to discuss a realistic approach, a use case from the automation domain, specifically from the cold rolling mill industry, is considered and discussed in this chapter.

7.1 Demonstrator - SMS-SEMAG Cold Rolling Mill

In the implementation, the following use-case from SMS-SEMAG cold rolling mill is considered. The SMS-SEMAG provides a demonstrator on which a simulation of the full plant runs. The demonstrator comprises the following servers:

- Industrial PC (IPC) for Profibus connection
- IPC for physics simulation
- Embedded PC
- G9 server for MATLAB model

Figures 7.1 and 7.2 show the demonstrator construction and the network plan respectively.

The SMS Group GmbH carries out a virtual commissioning in an integration test before commissioning a real system. All relevant control, regulation and technology functions of the plant are simulated by means of a hybrid simulator and the signals are transmitted to the connected Level 1 systems via Fieldbus. By setting up the real equipment (control station with HMI clients, on-site consoles), the virtual system can be operated completely with all automation-relevant functions and devices. The hybrid simulator consists mainly of the three components:

- I/O: The process image of the plant generated by the simulation is provided to the automation via emulated Fieldbus systems (Profibus, Profinet, EtherCAT).
- Dynamics and technology: Control and technological functions through mathematical relationships or differential equation systems can be described are mapped here with corresponding tools in real time. They are parallelized and distributed for performance reasons on several processes or computers.

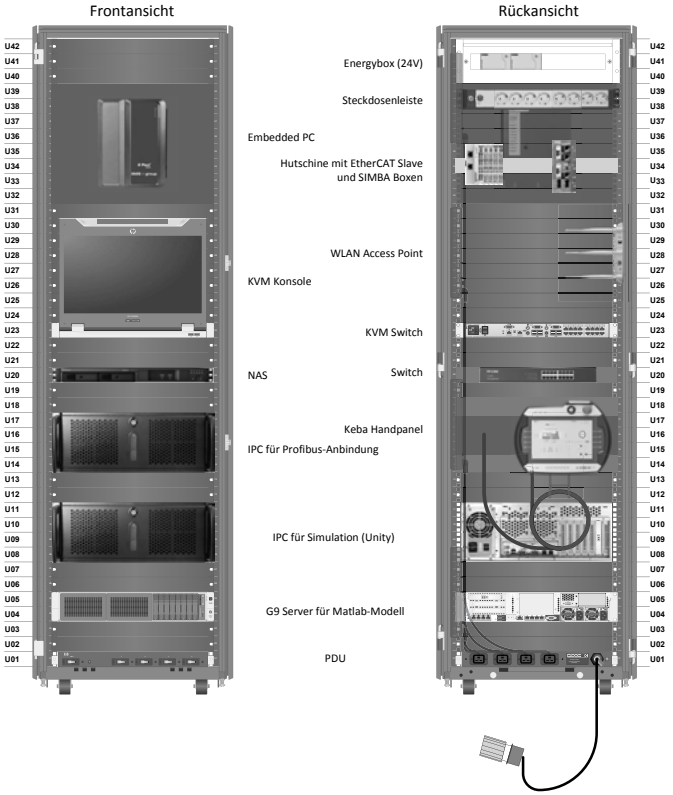


Figure 7.1: SMS-Demonstrator Racks

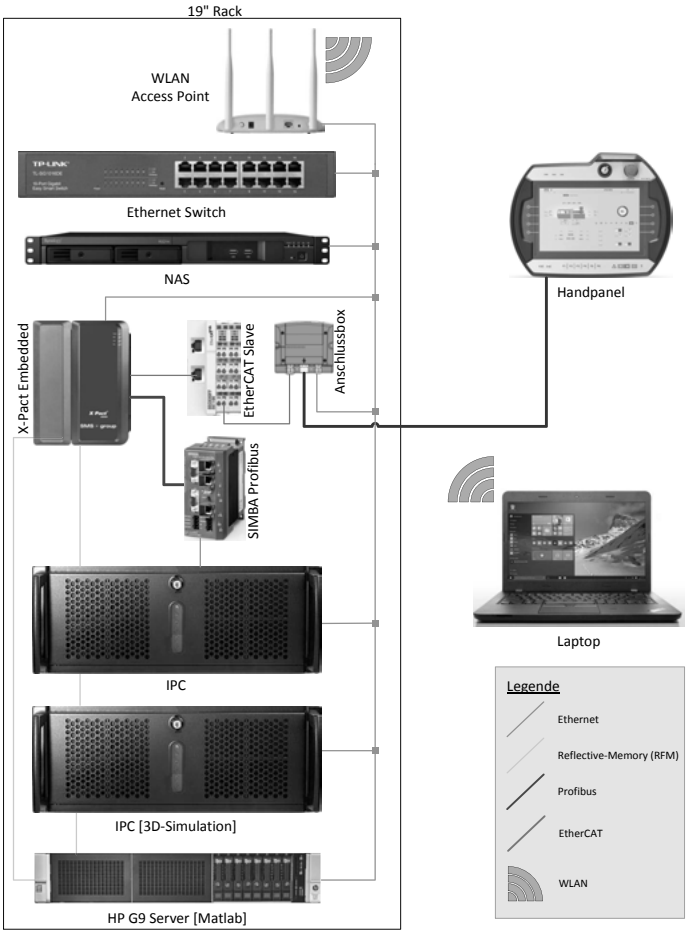


Figure 7.2: SMS-Demonstrator Network Plan

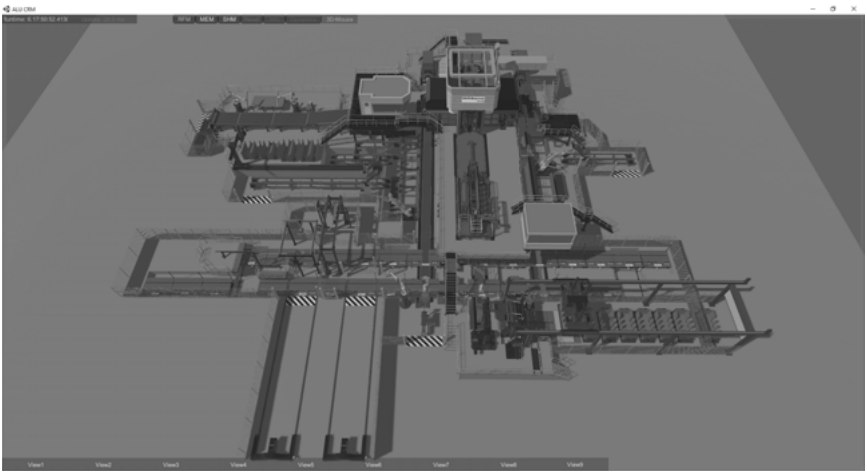


Figure 7.3: SMS Cold Rolling Mill as simulated by the Demonstrator

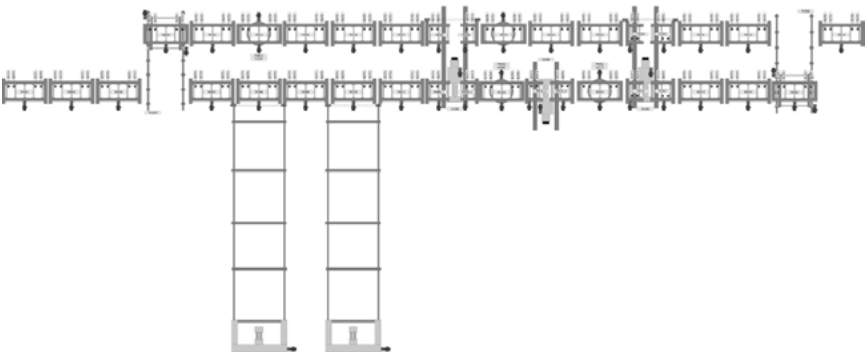


Figure 7.4: Full Blueprint of the SMS Cold Rolling Mill simulated by the Demonstrator

- Production and process simulation: The interactive 3D plant model uses an integrated physics engine that is based on real design data and maps the material flow throughout the plant. The plant is visualized in a manner similar to the view of an operator on the control platform on his plant.

The hybrid simulator is designed as a scalable system. Calculation steps of as small as $100\mu s$ are enabled in dynamic simulation. Basically, the simulation provides all system signals. Regulatory and control aspects are fully mapped, technological interrelations as far as necessary for the plausible operation of the automation are considered. The demonstrator provides a virtualization environment of the automation-related behavior of the plant and covers all production-relevant scenarios of an aluminum cold rolling mill which makes the creation of an adaptable scenario from a hardware perspective not only possible but also convenient, i.e., a new device can be created at any time or the topology can be changed dynamically. Operational dependencies and hazards as they occur during a real commissioning do not arise with this procedure. New automation functions or procedures can be efficiently tested without risk to man and machine or commercial risks. A simulation snapshot and blueprint of the full plant are shown in Fig. 7.3 and 7.4 respectively.

7.1.1 Devices

In this section, the X, Y, and Z axis are used to describe the motion convention in the horizontal axis (right and left), vertical axis (upwards and downwards) and the orthogonal perspective (in and out of the page) respectively. The directions correspond to the 2D plane figures and are used throughout the section. The demonstrator simulates the aluminum cold rolling mill plant of SMS. The plant is operated using many devices and divided into many sections. In the framework of testing the implemented work, a simplified example of the plant is taken into consideration as shown in Fig. 7.5. The simplified plant is a snippet of the full production line. However, the example includes a slight adjustment to include a third lane for better functionality illustration. The following devices are included in the example:

- Roller table: A roller table acts as a conveyor belt. This device is an active device, i.e., it has a motor that can move the palette in the X direction. The table has five sensors (light barriers for proximity detection) in which four are used to identify the palette position and one for coil presence detection.

- Shift table: Similar to a roller table with an additional actuation mode in the Y axis. The device has an additional motor that can move the shift table along with the carried palette (and also a coil) in the Y axis.

- Turntable: Similar to a roller table with an additional actuation mode that enables the device to rotate itself with the carried palette (and also coil) around the Z axis.

- Oven: Similar to a roller table with an addition production capability to heat the coil.

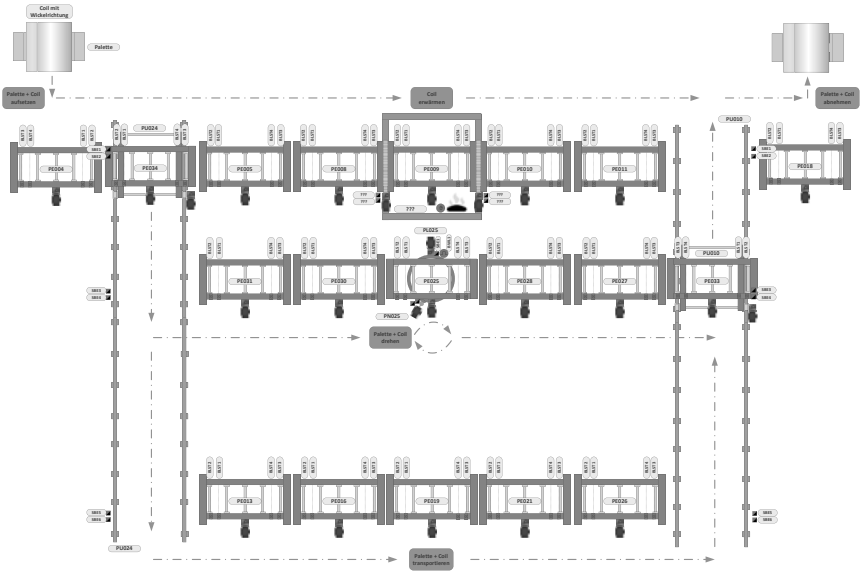


Figure 7.5: SMS-Demonstrator Extended Example

7.1.2 Single and Group Function Units (SFU and GFU)

The palette and the coil are physically passive, i.e., they do not possess an actuator and thus cannot physically move themselves. Each palette can carry an aluminum coil and can be transported by the plant logistical devices. Each device (e.g., roller tables and palettes) is controlled by a software component (a SCU and GCU respectively) in the runtime system. Although palettes are considered as passive devices, their corresponding software components of the palette take the lead in terms of orchestration between the active devices within the software architecture of the demonstrator. The palette GCU has information about the states of the real devices, e.g., the current position (i.e., the current roller table on which the palette is located), the names of the successor device (requires the assistance of a topology model), etc. For the purpose of control, a palette component can occupy a device and invoke a mode of operation. States of the devices are provided via a reflective memory from the simulation in real time for other applications (distributed control). The inner structure of the component follows the aforementioned structure in Sec. 2.5.4. The Single and Group Function Units (SFU and GFU) represent the aggregation of the entity and its corresponding control logic on device and orchestrator level respectively

Coil

The coil element represents the product entity in the plant. A procedure component is created to represent that are performed upon it. Each procedure component carries a

site recipe for the procedures to be executed on it. In the implementation, the recipe is modeled with Business Process Model and Notation (BPMN) using Camunda. The recipe specifies the required procedures on an abstract level.

Palette

A GCU is instantiated for each booked palette in the system. The GCU task is to orchestrate the orders to the devices SCU. Additionally, the GCU utilizes an occupation automaton to ensure safety and avoid collisions as well as commands overwriting from other devices.

Logistical Devices

Each devices has a SCU. The SCU is responsible for controlling the devices via the implemented logic. The component for each device can switch the motors on and off via a driver. The hierarchy and relationship between the different levels are shown in Fig. 7.6.

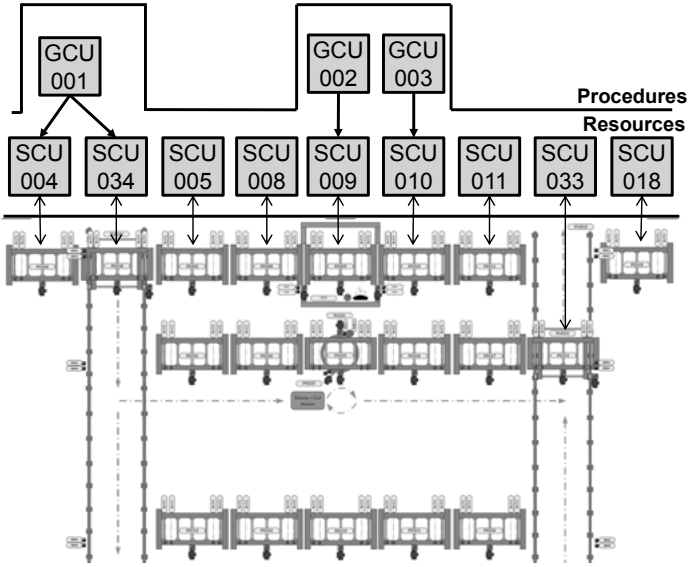


Figure 7.6: Procedures and Resources

7.2 Load Balancing

In order to distribute the load, two methodologies are utilized. The first approach uses decentralized algorithms to balance the loads and optimize the resources utilization in the network. Alternatively, an agent based method can be utilized instead. The objective of

the agent based method is to distribute the software components according to the flow of the product.

8 Implementation - Reality Approach

As shown in the previous chapter, in a practical example, the load is found in heterogeneously sized discrete packets. Fig. 8.1 shows a network example of the components assignment in a network. The figure illustrates the different layers:

- Device level: Where the process components, i.e., field devices, are found. In the presented use-case, the devices from the demonstrator (e.g., roller table).
- Control Level - Process Near Components (PNC): On this layer, the automation systems are found which performs the control logic of the process components.
- Server Edge Cloud: This layer provides the edge computing paradigm. Additional resources and services can be found on this layer.
- External Cloud: This layer comprises the components that provide an interface for the services and components in the layers underneath.
- Client: Where all client applications that interact with the system are found.

The objective of optimal distribution of components in the network can be approached from a resource (load distribution) or a component (agents systems based) perspective. In this chapter, a state of the art decentral algorithm is proposed that distributes the load using the resources approach. Furthermore, a system agents algorithm that solves the problem from an agent system perspective is shown.

8.1 Decentral Algorithm (Resources Perspective)

The decentral algorithm is inspired by the gained knowledge from the analytical and empirical models. The proposed algorithm uses a decentral method to balance the load. Decentral decision making behavior provides the following advantages:

- No single point of failures since the logic is executed in a distributed manner.
- No a prior knowledge of the network topology which provides better adaptivity readiness.
- No additional computation resources to execute the distribution optimizing is required. The load balancing computations are carried out on all the nodes which distributes and reduces the total overhead.

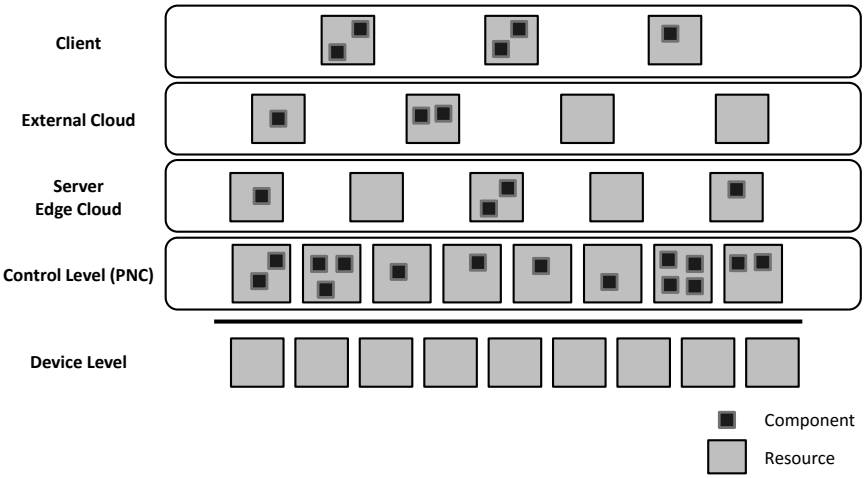


Figure 8.1: Components Assignment to the different Resources

8.1.1 Preliminaries

In this section, the fundamentals of the decentralized algorithm concept in the framework of the dissertation are explained. The algorithm concept employs a distributed intelligent behavior. Each node acts as a decentral independent entity in a self organized system. The main criteria of this behavior are:

- Only local information models can be considered. Information is retrieved through direct neighbors communication only.
- Decentralized decision taking, i.e., each node is responsible for its own decision taking and no central point influences the decision taking process.
- Each node relies solely on its own logic. The results of the individual nodes on a local scale propagate to form a successful collective behavior which can be seen in the results on the global scale.

8.1.2 Objective

The objective set for the algorithm is to balance the load amongst the available nodes in the network. However, defining an absolute value of the balanced load is not possible in case the nodes have different capacities. As a result, a generic objective of balancing the utilization percentage of the node resources is considered instead. In Sec. 3.4.1, the different optimization and constraints criteria are presented and demonstrated in Fig. 3.6. The presented constraints must be satisfied regardless of the utilized approach. Whereas, according to the approach, different optimization criteria can be considered. In the decentralized algorithm approach, the following optimization criteria are possible:

- Balanced utilization of resources on the controllers
- Redundancy of application

8.1.3 The BRAD Algorithm - Mechanism of Operation

As previously mentioned, the algorithm functions in a decentral manner. Each node is equipped with a discovery component that enables it to identify neighboring nodes. The algorithm operates in four synchronous cycles: “Broadcast”, “Request”, “Accept” and “Deploy”, hence the name “BRAD”.

Broadcast

The optimization criteria used by the nodes is the Total Square Error (TSE). In the first (Broadcast) phase, each node will inform its neighbors with its current load, capacity and system information. This information is used at each node to construct a local information model of the neighboring nodes. Furthermore, the information is used to compute the mean utilization value and the TSE consecutively. Thus establishing which nodes in the local neighborhood are above and below the mean value.

Request

In the second (Request) phase, load transfer requests between the nodes are exchanged. The load transfer request comply to the following six conditions:

1. Each node can only send and receive one packet at a time.
2. Requests are only sent from nodes above the mean value requesting a load transfer to nodes below the mean value and not vice versa (sender initiation).
3. The biggest load packet is chosen that decreases the TSE. If such a load packet does not exist, no request is sent out.
4. The request is sent to the best candidate node, i.e., the node with the highest negative deviation to the mean, as well as satisfies the requirements and boundary conditions, e.g., communication QoS and required Node to Device (N2D) topologies.
5. The maximum capacities of the node resources are considered. The request for a load transmission should not overload the receiving node.
6. In case more than one node satisfies the aforementioned conditions, a minimalistic choice is done, i.e., the candidate node with the least features (e.g., communication interfaces) that fulfills the requirements so that other candidate nodes with more features are spared for other requests.

The aforementioned conditions ensure the maximum gain per each execution step to optimize the TSE. Furthermore, they ensure the stability and convergence in the network regardless of the network topology. This can be seen from the derivation presented in Sec. 6.4.2- Stability. Ensuring stability on a micro scale (local neighborhood) ensures a stability on a macro scale (full network) recursively.

Accept

Similar to the previous phase, in the third (Acceptance) phase, a TSE optimization is performed. However, on the receiver's side, each node evaluates the requests sent by other nodes. The request with the highest gain is picked and an acceptance notice is thus sent to the corresponding node.

In the accept phase, the received requests are evaluated and a similar optimization as in the request phase is performed, however, to pick a load to receive and not to send. The best request that can minimize the TSE in the neighborhood is picked.

Deploy

Upon notification from the receiver node, the sender node prepares the corresponding component to be sent (serializes the data). In the fourth (Deploy) phase, the files are sent to the receiver node and the control is handed over.

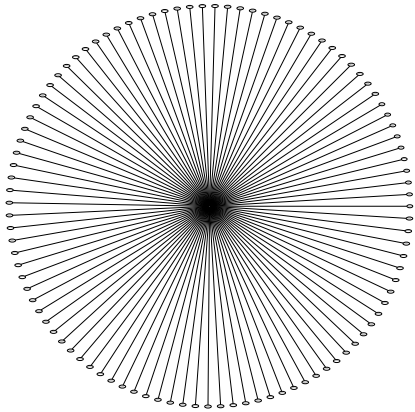
8.1.4 Simulation Assessment

Setup

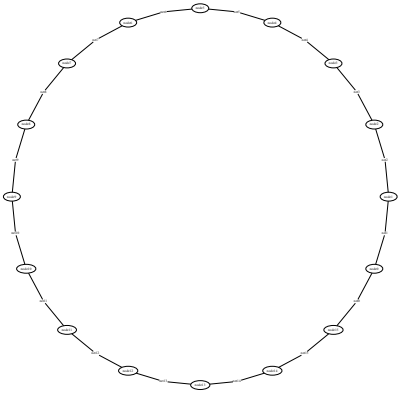
In order to assess the performance of the proposed algorithm in different situations, different networks topologies are considered. In the framework of the performance testing, the algorithm is tested in full mesh topologies with sizes of {10, 16, 25, 50, 100} nodes. Furthermore, assessments are conducted on Line, Ring, and $K_n - K_n$ topologies with 16 nodes to ensure functionality in other topologies. Fig. 8.2 shows some of the topologies that are used in the assessment.

Simulation and KPIs

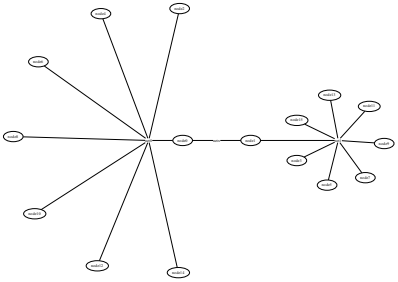
The proposed algorithm is simulated to ensure functionality and stability. Furthermore, the KPIs proposed in Chapter 6 are evaluated. Two scenarios were used to simulate different initial conditions. In the first scenario, all the nodes were given initial loads and the performance was measured until convergence, i.e., no longer load transfer operations were performed. Fig. 8.3 and 8.4 show the simulations of the load convergence and the total squared error progressions with time respectively. In the second scenario, in order to simulate the situations where spontaneous loads are placed at a node, 20% of the nodes in the network were given initial loads and the performance of the algorithm was measured. Fig. 8.5 and 8.6 show the simulations of the load convergence and the total squared error progressions with time respectively with initial load placement at 20% of the nodes. Tables 8.1 and 8.2 show the quantitative and qualitative KPIs respectively.



(a) Full Mesh - 100 Nodes



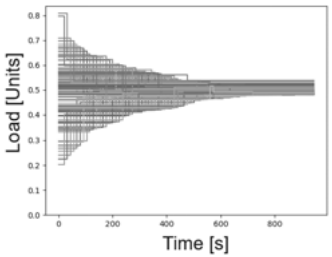
(b) Ring - 16 Nodes



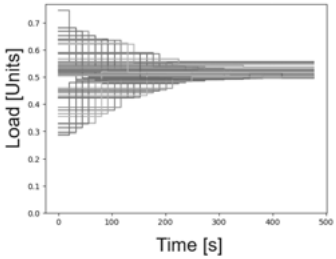
(c) Kn - Kn - 16 Nodes

(d) Line - 16 Nodes

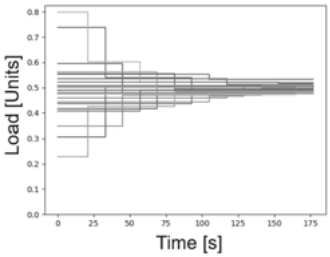
Figure 8.2: Example Topologies of the Simulated Networks



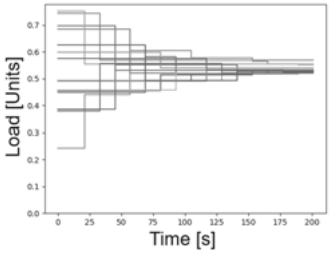
(a) Full Mesh - Initial Load:All - 100 Nodes



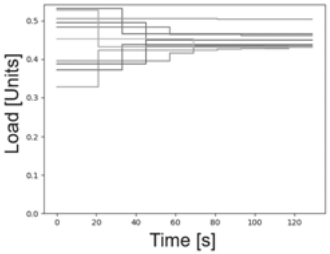
(b) Full Mesh - Initial Load:All - 50 Nodes



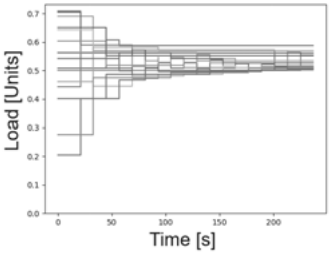
(c) Full Mesh - Initial Load:All - 25 Nodes



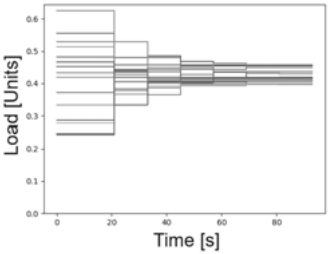
(d) Full Mesh - Initial Load:All - 16 Nodes



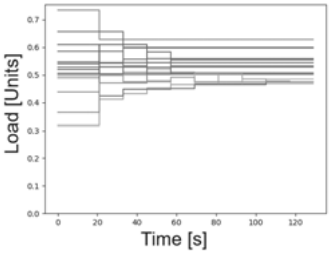
(e) Full Mesh - Initial Load:All - 10 Nodes



(f) Kn - Kn - Initial Load:All - 16 Nodes

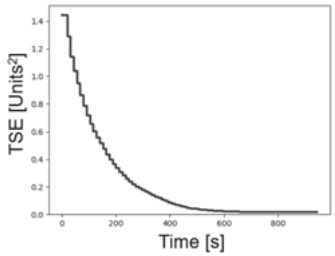


(g) Ring - Initial Load:All - 16 Nodes

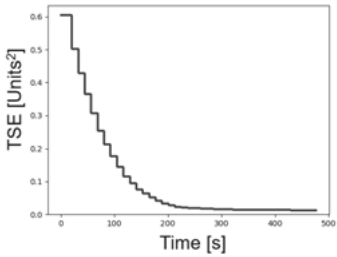


(h) Line - Initial Load:All - 16 Nodes

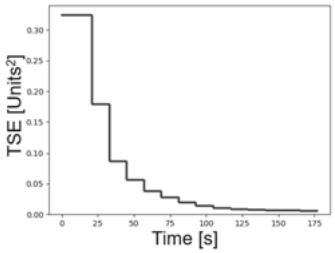
Figure 8.3: Simulation Results: Load Distribution Time Line with Initial Load placed at all Nodes



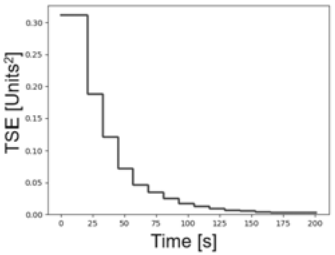
(a) Full Mesh - Initial Load:All - 100 Nodes



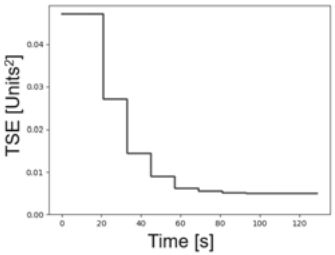
(b) Full Mesh - Initial Load:All - 50 Nodes



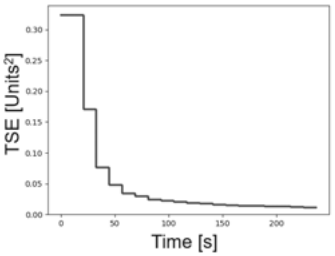
(c) Full Mesh - Initial Load:All - 25 Nodes



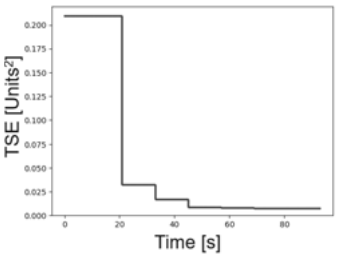
(d) Full Mesh - Initial Load:All - 16 Nodes



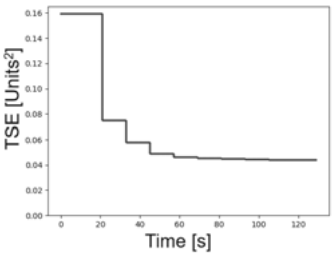
(e) Full Mesh - Initial Load:All - 10 Nodes



(f) Kn - Kn - Initial Load:All - 16 Nodes

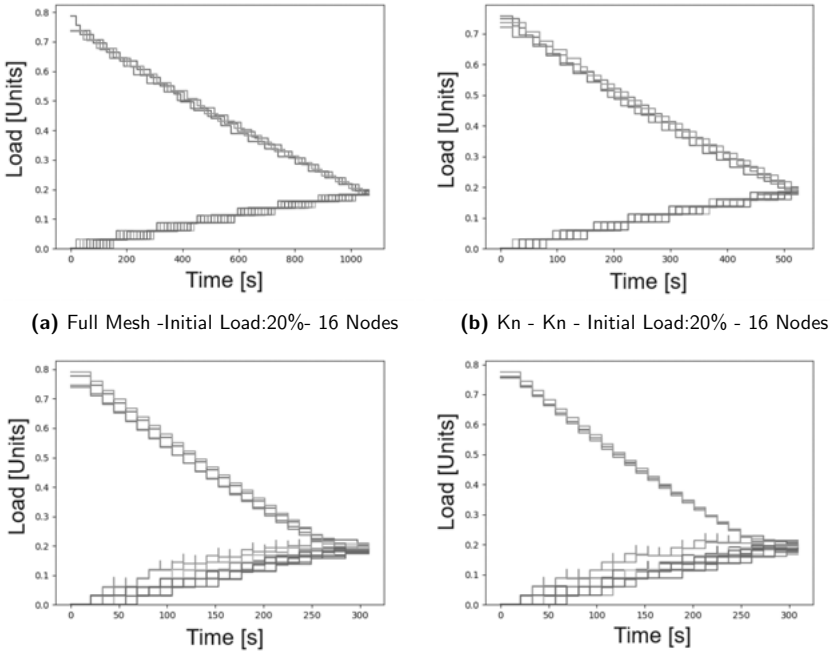


(g) Ring - Initial Load:All - 16 Nodes



(h) Line - Initial Load:All - 16 Nodes

Figure 8.4: Simulation Results: TSE Time Line with Initial Load placed at all Nodes



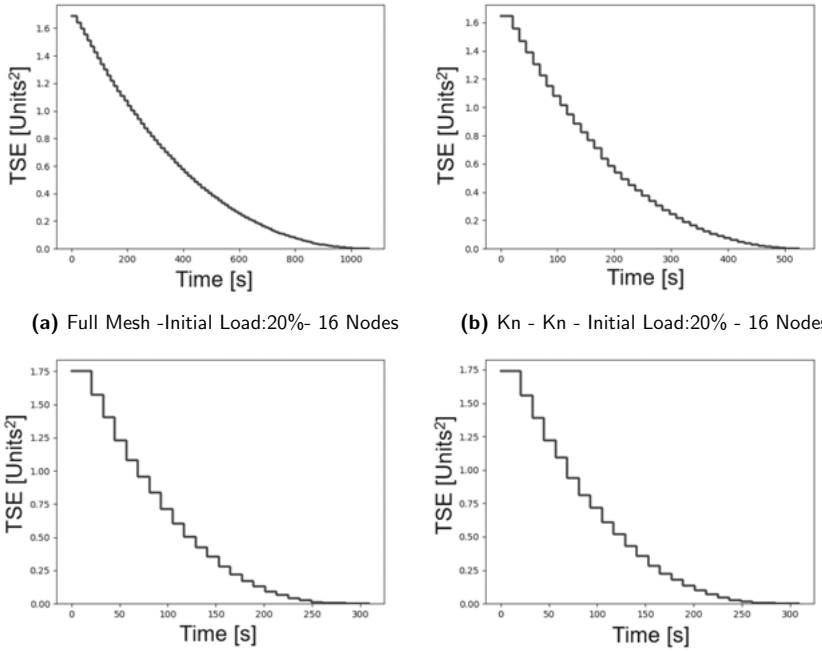
(a) Full Mesh -Initial Load:20%- 16 Nodes

(b) Kn - Kn - Initial Load:20% - 16 Nodes

(c) Ring - Initial Load:20% - 16 Nodes

(d) Line - Initial Load:20% - 16 Nodes

Figure 8.5: Simulation Results: Load Distribution Time Line with Initial Load placed at 20% of the Nodes



(a) Full Mesh -Initial Load:20%- 16 Nodes

(b) Kn - Kn - Initial Load:20% - 16 Nodes

(c) Ring - Initial Load:20% - 16 Nodes

(d) Line - Initial Load:20% - 16 Nodes

Figure 8.6: Simulation Results: TSE Time Line with Initial Load placed at 20% of the Nodes

Table 8.1: Quantitative Performance Assessment

Network			Time		Convergence					
Topology	n	Initial Load	Settling Time	Time Steps	$TSE^{(t_0)}$	$TSE^{(t_{25})}$	$TSE^{(t_{50})}$	$TSE^{(t_{75})}$	$TSE^{(t_{100})}$	DSCP
Full Mesh	10	All	129.02	10	0.04706	0.02708	0.00612	0.00495	0.00491	-0.00057
Full Mesh	16	All	201.02	16	0.31168	0.07228	0.01701	0.00511	0.00288	-0.00173
Full Mesh	25	All	177.02	14	0.32446	0.08652	0.01934	0.00713	0.00586	-0.00073
Full Mesh	50	All	477.02	39	0.60510	0.11464	0.01951	0.01371	0.01319	-0.00034
Full Mesh	100	All	945.02	78	1.44408	0.28253	0.04762	0.01761	0.01692	-0.00021
Full Mesh	16	20%	1065.02	88	1.68773	0.87577	0.35025	0.08075	0.00066	-0.00861
$K_n - K_n$	16	All	237.02	19	0.32412	0.03376	0.01887	0.01368	0.01133	-0.00180
$K_n - K_n$	16	20%	525.02	43	1.64542	0.88568	0.33931	0.07379	0.00100	-0.00837
Line	16	All	129.02	10	0.15953	0.07466	0.04597	0.04418	0.04398	-0.00064
Line	16	20%	309.02	25	1.73895	0.94056	0.28474	0.04563	0.00225	-0.00894
Ring	16	All	93.02	7	0.20971	0.03218	0.00870	0.00720	0.00712	-0.00116
Ring	16	20%	321.02	26	1.75120	0.95233	0.28587	0.02413	0.00157	-0.00902

Table 8.2: Qualitative Performance Assessment

KPI	Evaluation
Determinism	Yes
Initiation	Sender
Model Stability	Marginal
Repeatability	Yes

8.2 Agents Systems Approach (Components Perspective)

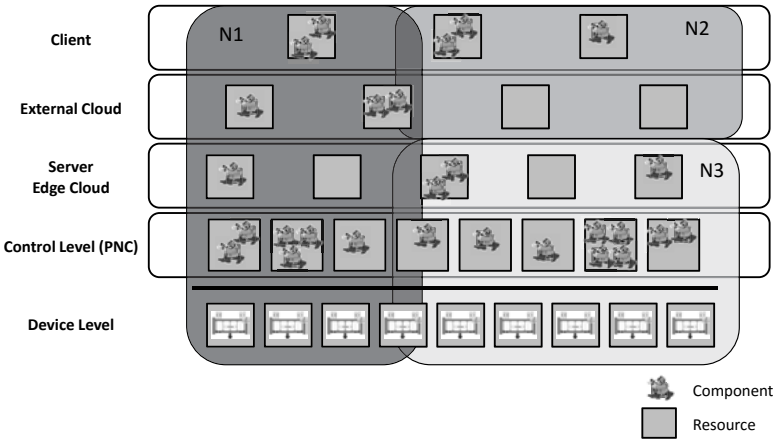


Figure 8.7: Agent System Approach Concept Mapped on the Use-Case

In order to tackle the load distribution problem, in the previous section, a resource perspective is considered. The main objective is to distribute the load evenly amongst the available resources. However, this perspective has a limited access to certain information that can provide an insight to other loads, e.g., communication loads and component cohesion, or other properties like security and safety in a network. On a process control level, a component can be considered as an agent, due to the sufficient autonomy it has. Considering the component as an agent provides a new perspective to the problem solution. Furthermore, the relationship between the entities in the information world (e.g., process control components and procedures) and entities in the physical world (e.g., devices and product) as well as the available boundary conditions can be investigated. An example of a boundary condition is illustrated in Fig. 8.7 with the gray scaled boxes labeled "N1", "N2" and "N3". The networks that connect the nodes and components together might contain different properties, e.g., communication QoS, security/safety grade, availability, etc. Examples for the optimization criteria that can be considered during load balancing:

- Reducing the communication load between computation nodes considering the information flow between the components.
- Optimizing the component placement according to the software component cohesion (component cohesion refers to the dependencies between the software components)
- Considering the network properties, e.g., safety and security aspects of the networks.
- Considering the product flow in the plant and using the shortest path which can give an insight into the communication between the entities in the field and their corresponding process control units.

Fig. 8.7 demonstrates the concept using the use-case presented in Chapter 7. On the devices level, as an example, the roller tables of the SMS demonstrator are shown. Each component in this approach is considered as an agent, e.g., the GCUs, procedures, cloud services, etc. Important to realize, the SCUs are considered to be static, i.e., cannot be redeployed due to the time criticality constraints. On the other hand, the GCUs, procedure and other agents can be redeployed to optimize the communication overhead between the software components. For this reason, in this specific use-case, the logistics perspective plays a crucial role, since the GCUs represent the pallets. In other words, the palette dynamic physical location or specifically the palette flow through the plant reflects the communication overhead between the components. Considering the palette carrying the coil as a mobile agent and using the recipe for each palette, the procedures can be computed and an insight into the palette flow through the production time can be predicted. This information can be used to redeploy the GCUs and their procedures agents throughout the production. The product flow is considered as a scenario for the optimum component placement from the perspective of the agent system approach.

9 Scenario 1 - Decentralized Algorithm

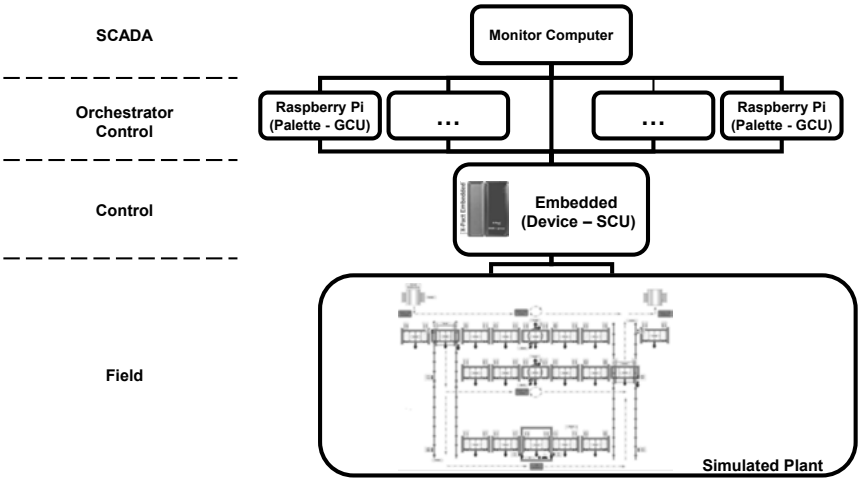


Figure 9.1: Use-Case Architecture

As previously explained, due to the critical realtime requirements, the SCUs are deployed to the embedded system. All other components, e.g., GCUs, can be freely distributed across the network assuming the QoS of the communication provided in the non-realtime communication channels is adequately sufficient for the system realtime requirements. In the presented scenario, the GCUs are used as the software components to demonstrate the redeployment concept.

9.1 Realization

The function blocks of the implementation and the connections between them are shown in Fig. 9.2. In the figure, the inputs are labeled with the corresponding signal number of the output connected to it. The ports marked with a black square at the edge indicate a communication over network, i.e., the signal is communicated from other nodes. In this section, the algorithm function blocks implementation is explained to demonstrate the logic flow.

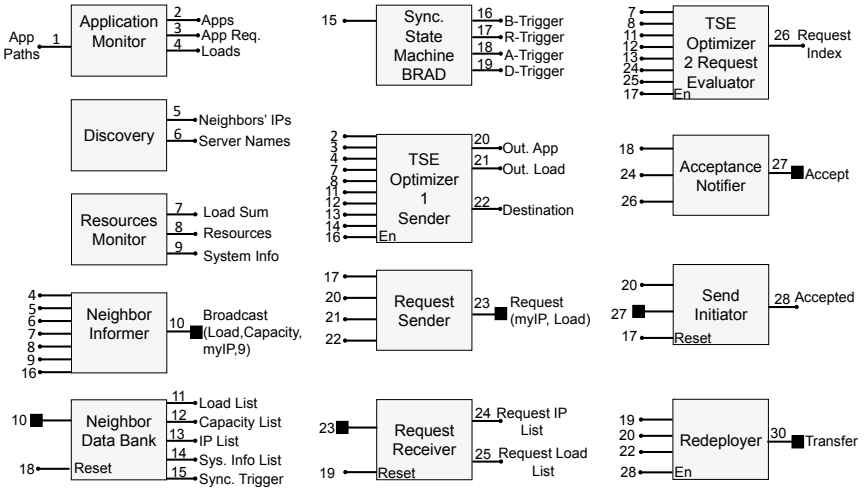


Figure 9.2: Algorithm Function Blocks

9.1.1 Sync. State Machine BRAD

The algorithm is executed in four synchronized phases. The BRAD state machine is the moderator of the four phases. An internal parameter is hard-coded that sets the time interval for the four phase (equal time for each phase). The hard-coded parameter is evaluated by estimating the maximum time needed to transmit the biggest load packet available in the network. This parameter ensures a robust synchronization between the nodes in the different phases. The state machine is reset as soon as any broadcast message is sent from a neighbor. The resetting strategy corrects any asynchronicity that might occur. A dwell time upon algorithm initiation is awaited to ensure synchronization.

9.1.2 Application Monitor

The application monitor component measures the load by monitoring the application under the given paths provided at the input. Only the applications listed under the given paths are considered by the load balancing algorithm, i.e., the ones the algorithm is allowed to redeploy. The objective of the monitor component is to list the application, their corresponding requirements and loads.

9.1.3 Node to Node (N2N) Discovery

N2N discovery is essential to list the possible redeployment destination candidates. A library is implemented that uses mDNS protocol to discover the available ACPLT/RTE in the network. Upon starting the ACPLT/RTE, the mDNSRegisterer component is initialized which registers the OV server instance using Bonjour protocol so that it can be discovered by other nodes. Three components deliver the discovery functionality in the

implementation:

ACPLT/RTE Server Discovery

This component lists all the discovered servers on local network. Each entry is a tab-delimited string and structured as follows: DNS fullname; DNS interface; OV host; OV Port; OV servername as shown in Fig. 9.3. Another point to be noted in the figure is the ip list discovered. In the list, there are two ip addresses shown, where one is repeated four times. This occurs due to self discovering at different interfaces namely in Ethernet, wireless network, virtual adapter of the internal communication in the RTE and virtual adapter of the system. The redundant node entries are filtered when considering a neighbor for load distribution.

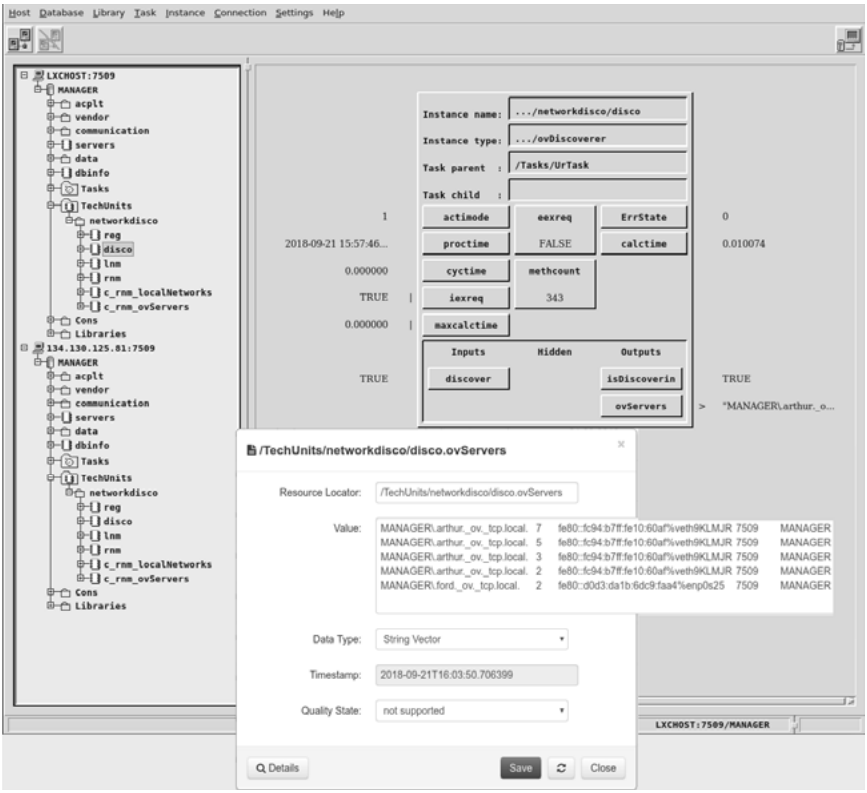


Figure 9.3: Discovered ACPLT/RTE Servers

Remote and Local Network Discovery

Figures 9.4 and 9.5 show the output of the local and remote discovery components respectively. The figures shown present the discovery using the network that is used by the infrastructure of the University. The components show the results outputted when testing with two computers where both are connected via local network and one of them is connected to the “eduroam” remote network.

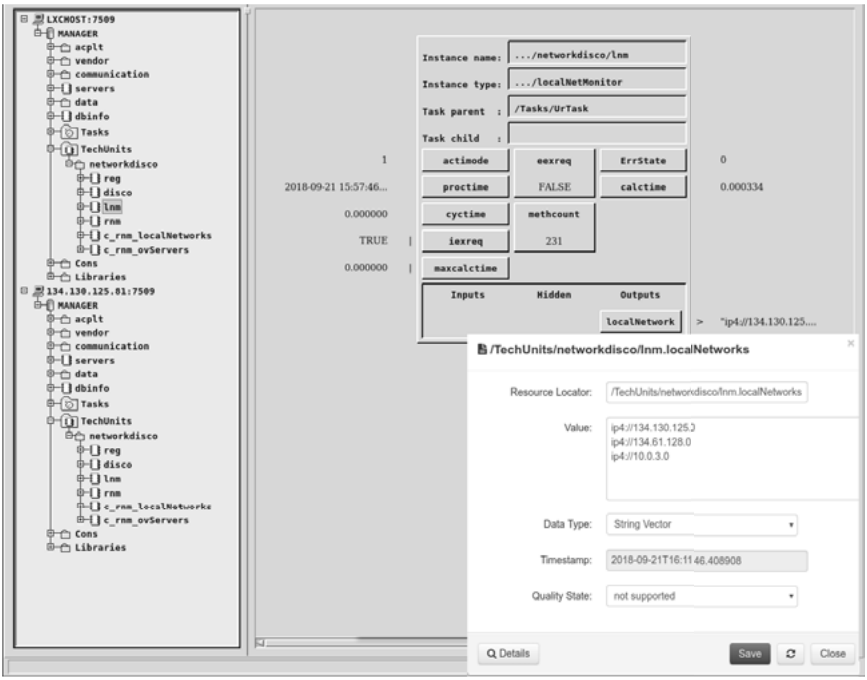


Figure 9.4: Discovered Local Networks

9.1.4 Device Resources Monitoring

At each node, a resources monitor component is run to measure the utilized and the available resources. The following information attributes are provided:

- fieldDevices: a static list of the connected field devices to the node (N2D) topology that is manually inputted.
- sysOS: operating system of the node (linux, nt, etc.).
- sysOSVersion: Operating system (kernel) version
- cpuType: Model name of the system’s CPU.

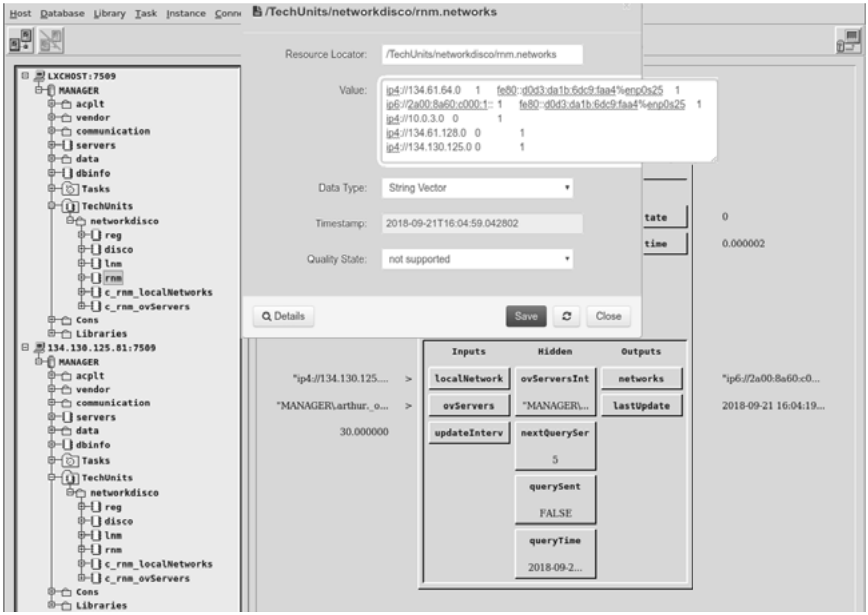


Figure 9.5: Discovered Remote Networks

- cpuUsage: Average CPU usage (computation time/total time) during last update interval
- memSize: System memory in Kibibyte
- memUsed: Used memory in Kibibyte (does not include buffers and caches on Linux operation system)
- ovArch: OV CPU architecture (i386, AMD64, ARM, ARM64)
- ovDBSize: Maximum allowed size of the OV database in Kibibyte that can be used
- ovDBUsed: Current utilized size of the OV database in Kibibyte
- ovLibs: List of currently loaded OV libraries
- ovFbUrCycTime: Cycle time of the FB-UrTask in seconds
- ovFBUrCalcTime: Calculation time of the FB-UrTask in seconds
- cpuLastTicks: Saved number of total CPU time ticks at last measurement
- cpuLastIdleTime: Saved number of CPU idle time ticks at last measurement

The resources monitor component is used during the destination node search process. A snapshot of the implementation is shown in Fig. 9.6.

9.1.5 Neighbor Informer

The neighbor informer component broadcasts information about the node to the neighbors. The broadcast message comprises information about: the total load, resource capacity of the node, its ip and system information, e.g., operating system, available libraries, available hardware, etc. The system information is used during the load transfer requests ensuring availability for application requirements. The informer block starts operating upon receiving a “Broadcast-State-Trigger”.

9.1.6 Neighbor Data Bank

This component archives all the broadcast messages of other nodes. As soon as the first entry is pushed in the list, a flag is set. Consequently, the BRAD state machine is reseted to the broadcast phase firing the “Broadcast-State-Trigger”. The block is reseted and lists are cleared as soon as the “Acceptance-State-Trigger” is fired.

9.1.7 TSE Optimizer - Sender End

The TSE optimizer receives internal information about the current (potential candidate) application for deployment as well as its requirements. Furthermore, it receives neighbors’ information from the data bank component. It performs a local optimization to determine the best node candidate and the optimum load packet to be transmitted to minimize the TSE in the neighborhood as mentioned in Sec. 8.1.3. This component provides the essential information to send a load transfer request.

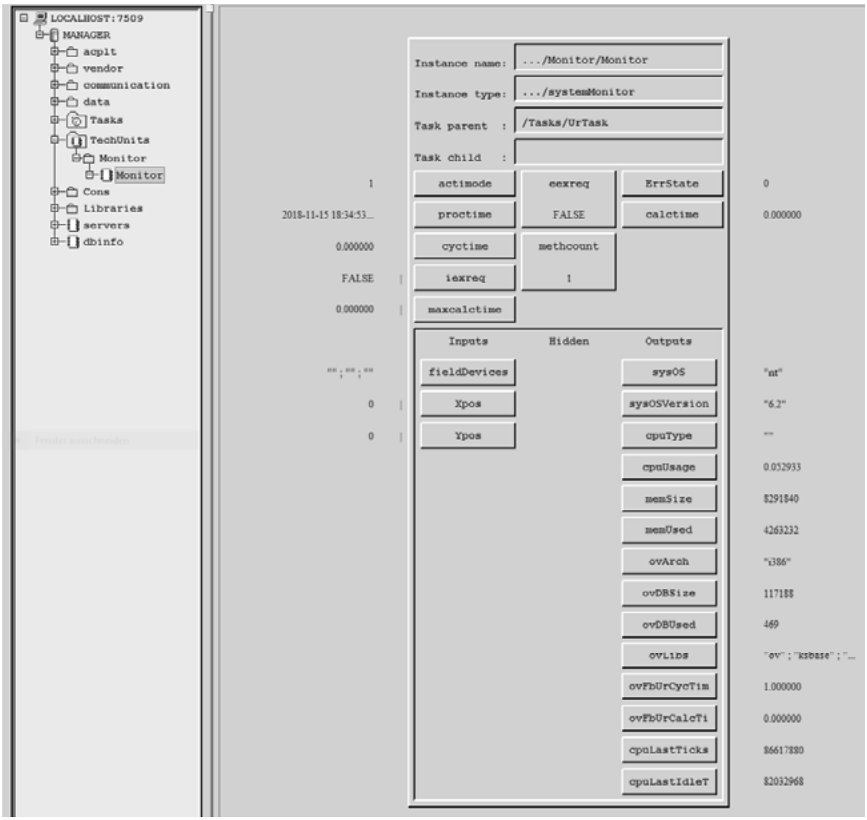


Figure 9.6: Snapshot of the Resources Monitor in ACPLT/RTE

9.1.8 Request Sender

The request sender component packs the information provided by the sender TSE optimizer, constructs a request and sends it to the request receiver of the corresponding node upon receiving the “Request-State-Trigger”.

9.1.9 Request Receiver

This component is the inbox of the upcoming load transfer requests from other nodes. It uses the requests to construct two lists, IP list and load list and forwards them to the Receiver TSE optimizer for evaluation. The lists are reseted by the “Deployment-State-Trigger”.

9.1.10 TSE Optimizer - Receiver End

As mentioned in Sec. 8.1.3, the requests are evaluated and the best candidate is accepted for a load transfer. The information about the picked request is forwarded to the acceptance notifier component.

9.1.11 Acceptance Notifier

Once the “Accept-State-Trigger” is fired, an acceptance notification is sent to the corresponding chosen node.

9.1.12 Send Initiator

The send initiator prepares the node for redeployment. The component provides the application path of the picked load allowing enough time prepare for the serialization.

9.1.13 Redeployer

Once the “Deployment-State-Trigger” is fired, the redeployment execution is initiated. Components that have states must be synchronized upon deployment. However, in the framework of the implementation, deployment is performed without synchronization. The logic flow of the redeployment execution is shown in Fig. 9.7. The components under the provided directory are serialized into a JSON String. A snapshot of the states is taken. During the snapshot capturing, the database is locked to avoid compromising the states of the components. Important to realize, this mechanism succeeds, i.e., recreating an identical component when loaded, only if the component is developed in a way such that all internal variables are made visible to the database layer (which must be considered in the component development stage), i.e., no internal variables are stored in C types and not communicated to the OV model variables. The database is then unlocked after the snapshot is captured. Before transmitting the JSON String to the destination node, an availability check for the libraries required is performed. Should one or more libraries be missing, a system check is performed to ensure compatibility before the libraries are transmitted. In case of incompatibility, the system aborts. Otherwise the libraries are serialized and consequently transmitted before the component. The essential libraries are

then loaded and the serialized JSON String of C1 is then transfered to the destination node. Upon successful transmission of the serialized JSON String, the loading component initially deserializes the JSON String. An identical copy of the structure is created at the destination node then C commands are used to perform override write operations on all variables ensuring an identical recreation of the captured component and preparing it for synchronization.

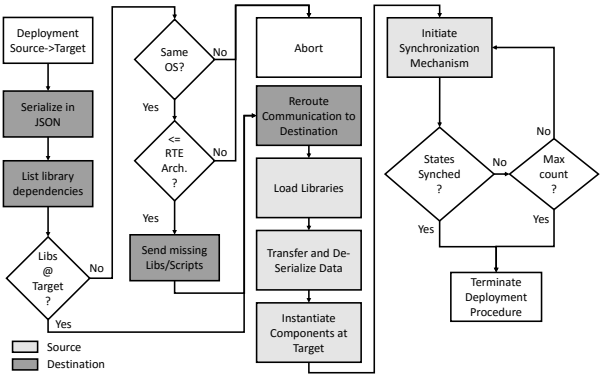


Figure 9.7: Logic Flow of the Component Migration

9.2 Performance Assessment

For the performance assessment, the following scenario has been performed to assess and validate the algorithm implementation

9.2.1 Setup

The setup is shown in Fig. 9.8.

Computation Nodes

For the purpose of providing resources for orchestration, three Raspberry-Pis (Raspis) and two PCs are used as computation nodes to test the presented adaptation concept. The Raspis are all connected via Ethernet communication whereas the PCs via WiFi.

Runtime Environment

The ACPLT/RTE is used as the runtime environment to program the logical control charts using IEC 61131 function blocks.

Communication

Ethernet and Wifi are used to connect the computation nodes. Furthermore, ACPLT/RTE legacy KS communication protocol is used for the communication between the nodes in the presented use-case.

Monitoring

The load history of the nodes and the algorithm performance are monitored using a computer terminal. The terminal executes a python script that polls the load statuses of the nodes.

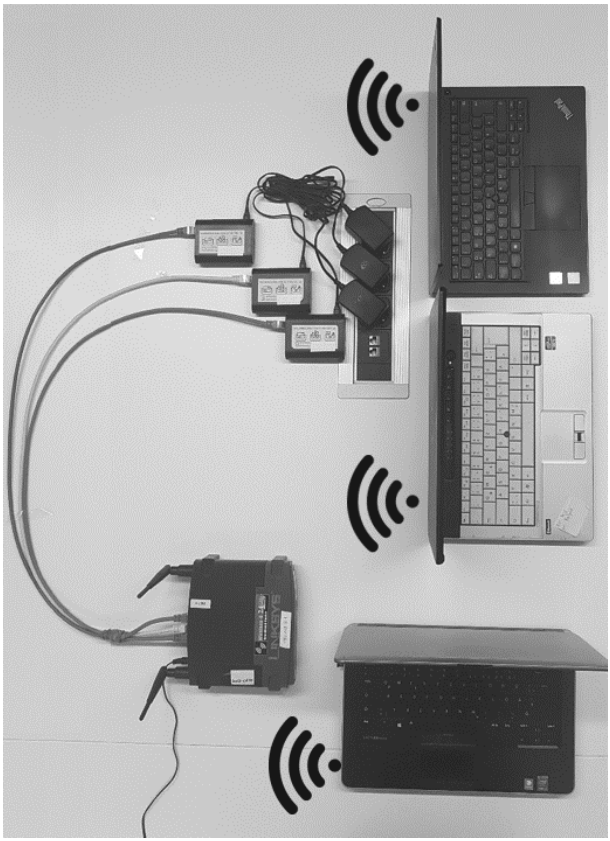


Figure 9.8: Setup of the Demonstrator

9.2.2 Scenario

The following bullet points refer describe the plot shown in Fig. 9.9

- Initially the network starts with four nodes (two PCs and two Raspis) for the load balancing network and a terminal to monitor the load status at each node.
- After all systems are started, an initial load is deployed to two of the Raspis namely Raspis 1 and 2 with load of 100 and 200 points respectively.
- The initial deployed load has a total of 300 load units and comprises the following GCUs arrangement: GCU1 (5X), GCU2 (8X) and GCU3 (3X)
- Table 9.1 is used to manually define the arbitrary loads assigned to the GCU components. Important to realize, the table shows the load value assigned to the GCU types and not instances, i.e., GCU1 is a type definition where instances are named after followed by a suffix. For example GCU1_56 is an instance of type GCU1 and costs 10 load points (cf. Table 9.1).

Table 9.1: Weights Assignment for Components

Component Name	Weight (units)
Palette 1 (GCU 1)	10
Palette 2 (GCU 2)	20
Palette 3 (GCU 3)	30

- The capacities of all nodes are considered equal (assigned as 200 units), i.e., Utilization percentage can be computed by normalizing the values with a factor of 2.
- The first broken line in Fig. 9.9 marks the event of the first convergence. At this instant, a new Raspi board with zero utilization is introduced into the network architecture providing resources for load distribution. Using the implemented discovery service, the Raspi boards are able to discover their neighbor IP addresses automatically and use the newly provided resources.
- Time step 300 marks the second load distribution convergence.
- The second broken line shown in the figure marks the event of deploying a new instant load of three GCUs of type 30 (i.e., weight 30 units each) to Raspi 2 node.

The algorithm showed successful operation in terms of discovering newly introduced nodes in the network as well as newly added load. At each disturbance event, the algorithm rebalanced the load amongst the nodes and converged to the optimum balanced load value. The results are shown in Fig. 9.9.

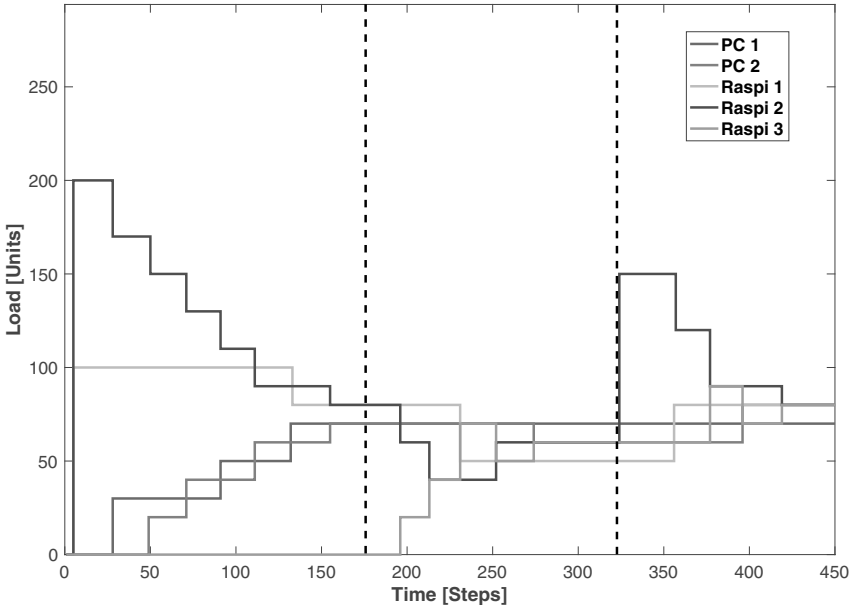


Figure 9.9: Load Distribution Timeline at the different Nodes

10 Scenario 2 - Agents System

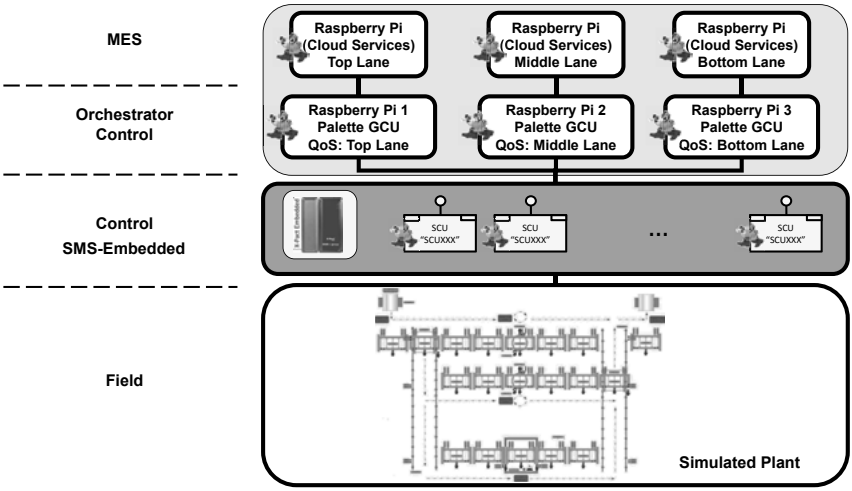


Figure 10.1: Use-Case Architecture

As shown in Fig. 10.1, similar to the previous scenario, the components of the devices are deployed to the embedded system. All other components (including cloud services) can be freely distributed across the network. For this purpose, Raspberry-Pis are used in the implementation to provide resources for the orchestration and MES layers. "Open-knowledge-driven manufacturing execution system" [50] can be seen as further potential examples of cloud services. As shown in the figure, each Raspi is directly connected to a cloud service providing it with a certain QoS for each lane. Having such an arrangement induces an important role for the physical position of the GCU in the plant (cf. Sec. 8.2). The current position of the pallet gives an insight into the communication load between the cloud services (MES layer) and the GCUs (orchestrator control layer).

10.1 Realization

In the realization, as a proof of concept, a recipe is written to be inputted to the algorithm and to compute the information required by the redeployment algorithm using the agents approach. The structure of the roller tables in the simulation is modified in order to better illustrate the functionalities of the presented algorithm. The modified plant is shown in Fig. 10.2. In the modified plant, the oven is placed at the mid-bottom layer. Two turntables

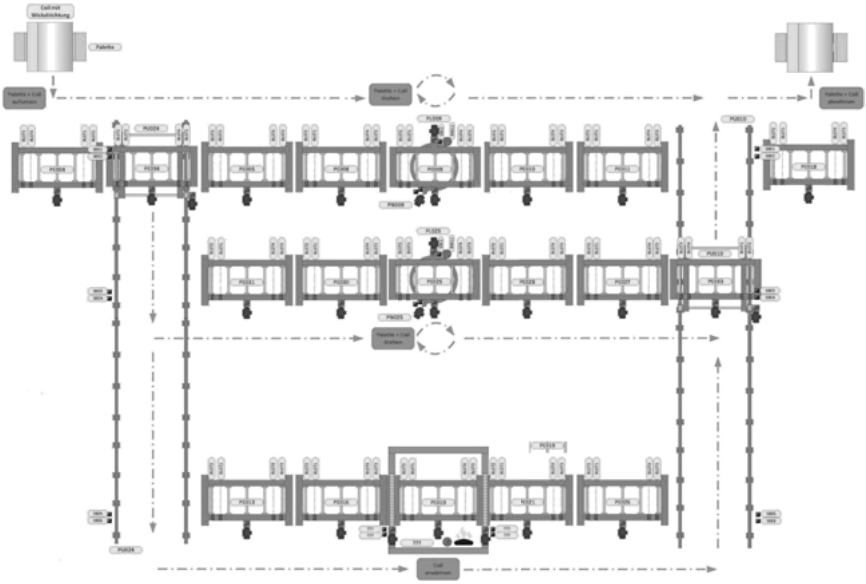


Figure 10.2: Modified Use-Case Architecture

are inserted in the mid-top and mid-middle layers. The recipe uses predefined keywords and symbols as defined in Table 10.1. An example recipe written in Camunda is shown in Fig. 10.3. The keyword "Storage" here is internally defined as an alias for the address of roller table "PE018" (roller table located in the top right corner). The recipe indicates that the pallet should travel from the top left roller table (current position), to the middle of the bottom lane, coil should be heated and then transported to the top right roller table.

Figure 10.3: Example Snapshot of a Recipe written in Camunda

The algorithm performs the handover as shown in Fig. 10.4. The figure shows the procedures executing agents at a current position of "PE034_1", i.e., shift table. The shift table connects all three lanes. However, the algorithm can anticipate the next location from the Production Flow Map (PFM). Hence, it synchronizes the agent with Node 3 (which has a better QoS to the bottom lane) instead of Node 2 preparing for the handover as shown in Fig. 10.4.

Table 10.1: Abstract Recipe Keywords

Keyword	Task	Arguments
Transport	Move coil/palette from the actual position to a given position or to an assigned name for a certain address	Exact address or assigned station name
&Capability	An '&' operator is used to indicate a capability, e.g., heat in this use-case (the keyword is abstract and is searched for in the production capabilities list). The nearest position (device) in the vicinity that pertains such a capability is retrieved and the palette is routed to it	Production capability
Position &Capability	An explicit identification of the device that pertains a production capability. Here the nearest device will not be looked up rather the palette will be driven to the explicitly specified device where the process will be executed	Production capability

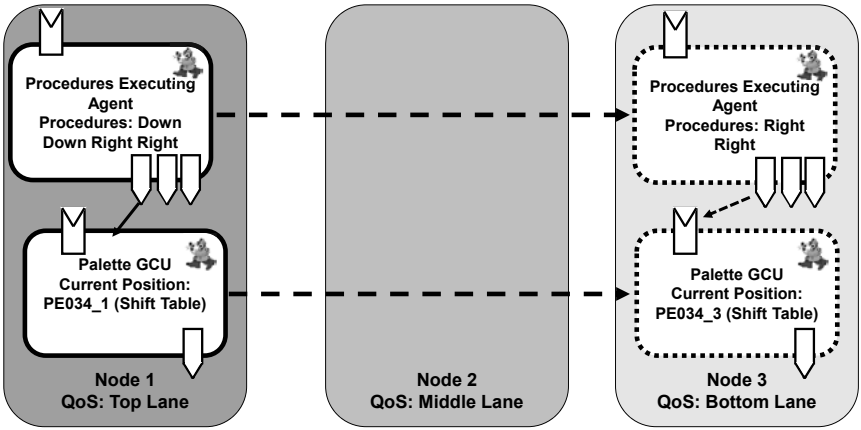


Figure 10.4: Agents Systems Architecture

Path Finder

The product flow planning is used as input data for the agents system approach. The product flow planning uses a given weighted graph that demonstrates the plant devices topology. Using Dijkstra algorithm [24], a route is planned and taken as initial indicative information for the GCU procedures' path. Using this information, a cost function can be formulated. The cost function demonstrates the communication load and QoS variation between the different computation nodes versus the planned route. Consequently, a plan for the redeployment of the GCU procedures can be constructed. The algorithm is implemented in such a way that a list of destinations can be inputted as a vector. As a result, the algorithm computes the optimal routes starting from the current position where the palette is located to the final destination via all the provided stopovers. In the input vector, a production capability must be preceded with an '&' operator to indicate that it is not an address name rather a capability. Furthermore, the production capabilities can be used as a destination, i.e., if "&HEAT" is written as an input, the algorithm searches for the closest destination that can perform this capability and computes the optimal route to it. Alternatively, an explicit destination address can be given for a capability with the '&' operator as a delimiter which indicates that the production capability must be performed at this exact destination, e.g., "PE019&HEAT". Important to realize, the path finder computes topology in vector form. Thus upon termination, the path finder maps the direction vectors on to a topology that is understood by the services.

10.1.1 Agent Load Balancing Algorithm

The agent load balancing algorithm uses two information models to perform the redeployment. Firstly, the algorithm uses the PFM computed by the Dijkstra algorithm as the drive for the redeployment. Secondly, a Quality Information Model (QIM) is fed to the algorithm. The QIM provides the load profile according to the agent location. The load is computed according to the application cohesion and QoS of the communication. The application cohesion indicates the distance between the procedures executing agent and the applications of the devices whom with it communicates. The algorithm monitors the current location state of the agent to be redeployed and compares it to the PFM. Using the PFM and QIM, the algorithm can anticipate a load optimization that can be executed in upcoming location states. Thus, the algorithm produces a copy of the procedures agent and the GCU and synchronizes it with the original one. Once the anticipated state is reached, the handover is performed.

10.2 Performance Assessment

In the following subsections, the setup, scenario and outputs are demonstrated as an assessment for the algorithm.

10.2.1 Setup

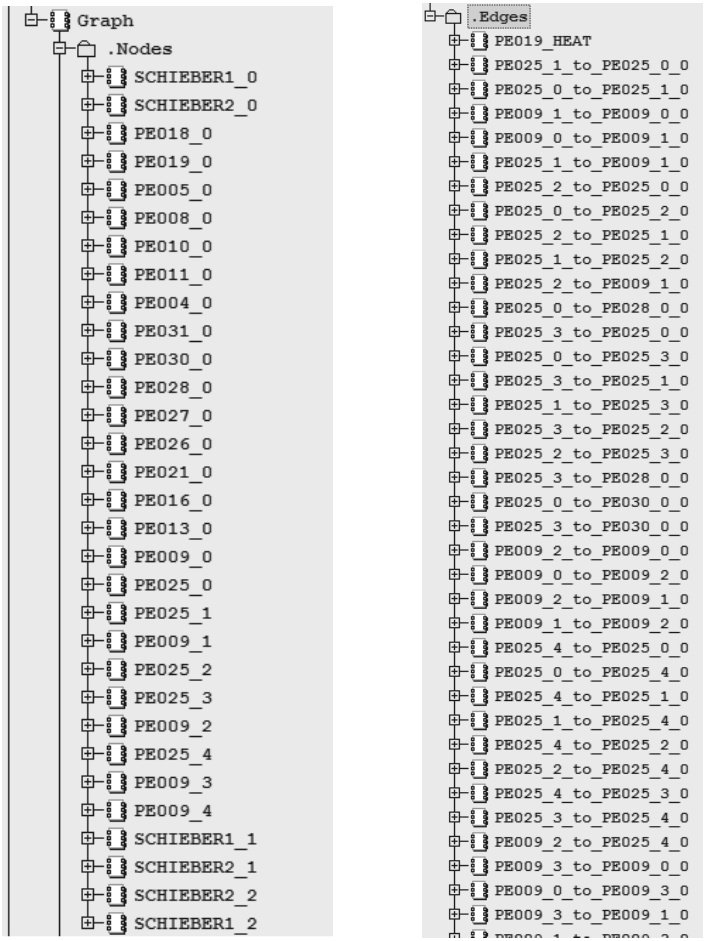
Same setup is used as in the load distribution scenario, i.e., Raspberry Pis are used as computation resources, ACPLT/RTE as the RTE, etc.

10.2.2 Scenario

- Initially the network starts with three Raspberry-Pi boards for the orchestration layer.
- Each board is connected to a simulated cloud service of a certain lane of the simulated plant.
- Being directly connected to the cloud services of a certain lane provides a better QoS to the corresponding lane.
- The GCUs are deployed randomly to an arbitrary board.
- There exist three function units (three palettes in the plant and their three corresponding operating GCUs).
- The three palettes are carrying out the same recipe.
- A coil should be transferred from the starting position (where the palette is located) to “PE009” (mid-top lane), then to the closest oven for heating, then transported to ’Storage’ position, i.e., “PE018” (top lane - right).
- Bypassing edges is not allowed, i.e., shift tables must perform a stopover at each lane.
- The route planning is performed according to a given cost model that depends solely on time and does not consider other factors.

Fig. 10.5 shows the nodes and edges of the graph. The topology mapping is evaluated as shown in Fig. 10.6. The figure divides the plane into 4 four quadrants which are used to map and discretized the vector directions to the defined topology convention used by the implemented services.

Fig. 10.7 shows a snapshot of the path finder implemented in ACPLT/RTE. The interpreted recipe is shown as ““PE009”, “&HEAT”, “PE018””, with a starting position at “PE004”. The palette should travel from that starting position at “PE004” to the turntable in the first row from the top, then to the closest position where the “HEAT” capability is available, perform the heat capability and then transport the palette to the last destination at the top right roller table “PE018”. The outputs of the pathfinder are shown in Fig. 10.8 and 10.9 with their respective clarifications in the caption.



(a) Node List **(b)** Edge List (not entirely shown)

Figure 10.5: Nodes and Edges of the Cost Model Graph

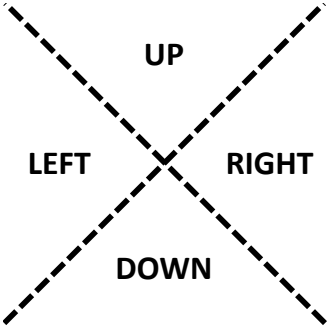


Figure 10.6: Discretization Scheme of the Vectors for Topology and Service Mapping

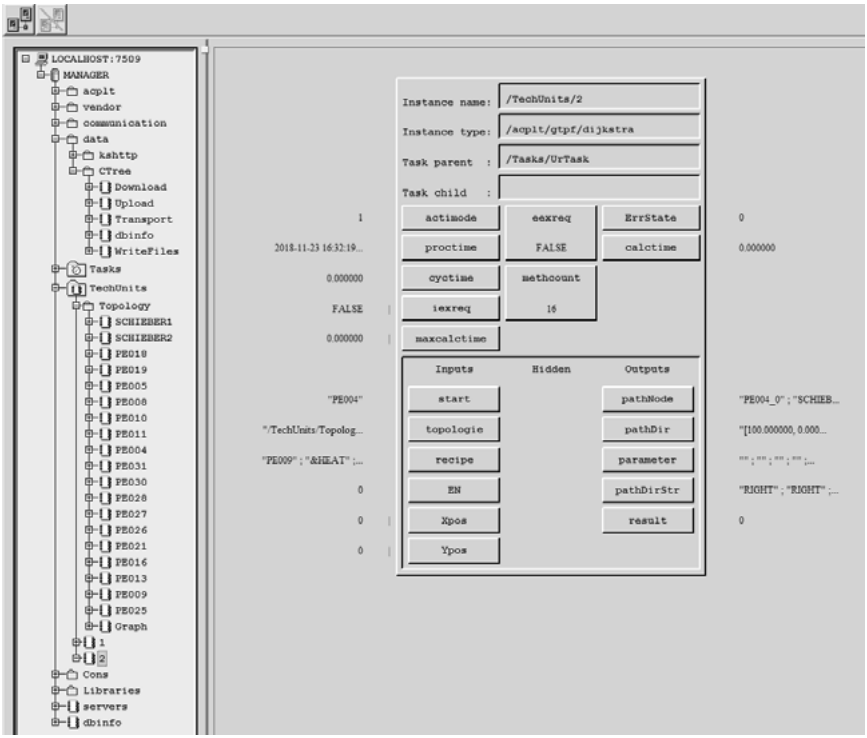


Figure 10.7: Snapshot of the Path Finder in ACPLT/RTE

Figure 10.8: Name of the Nodes in Planned Path

Figure 10.9: Mapped Topology for the Service Mapping Stage

11 Conclusion and Outlook

In this dissertation, a concept of adaptation in industrial automation is presented. The concept of adaptation is considered from different perspectives to provide a thorough analysis. Firstly, a control theory approach is utilized, where stability, system dynamics and response are analyzed. Additionally, a multi dimensional problem definition as well as system dynamic analysis are presented. Furthermore, a regression model is constructed to investigate hub spoke networks and optimize their Markov chain mixing time. Secondly, the system is analyzed with an empirical approach. In this approach, a benchmark and a list of KPIs are introduced to assess the performance of algorithms.

Although the previous approaches provide good results and an informative insight to the load balancing problem, a considerable reality gap still exists which hinders utilizing the achieved results directly. In order to overcome the reality gap, a realistic approach that uses an aluminum cold rolling mill use-case is investigated. As a result, two scenarios are demonstrated as a solution for the adaptation problem using redeployment. The first proposed solution utilizes a decentralized distribution algorithm. The algorithm balances the loads according to the overhead and the resources reserves in the computation nodes. The second proposed solution uses an agents systems approach which considers the production flow. Furthermore, two prototype implementations are realized and demonstrated to serve as a proof of concept.

11.1 Outlook

As an outlook for this work, the following points can be implemented:

11.1.1 Algorithm Enhancement

Currently, the implemented algorithm performs a TSE optimization to choose the load packet to send and also to receive. A fusion between the TSE approach and the approach presented in the analytic approach (continuous load) shown in Chapter 5 can be introduced. This can be done if a beforehand optimization of the load transfer coefficients α is performed. The values of the transfer coefficients can be considered by transferring more discrete load packets. In other words, the results of the analytical approach can be discretized and adapted such that they do not exceed the resulting value computed by the analytic approach. Performing transfer in such a manner allows transfers of more than one packet at a time and considers the convergence as well as the set optimization criteria simultaneously. However, in order to conduct such an analysis, the network topology has to either be known or a discovery mechanism to explore the full network topology must be initially performed. This enhancement can be rendered useless if the topology of the network is rapidly changing. Moreover, it can also be argued that such an enhancement can hinder its adaptability property.

11.1.2 Synchronization

As mentioned in Sec. 9.1.13, components that have states must be synchronized upon deployment. As an outlook for the implementation, a mechanism for the states synchronization can be implemented. The planned redeployment mechanism comprises four stages: communication rerouting, component migration, states synchronization and handover. Fig. 11.1, shows the required components to perform the four stages.

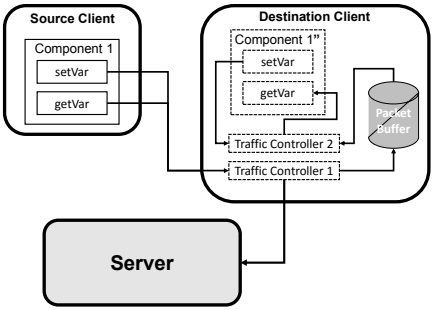


Figure 11.1: Redeployment Illustration with the involved Components

Communication Rerouting

The initial step performed in the communication rerouting stage is creating "Traffic Controller 1" (TC1) and a "Packet Buffer" (PB) at the destination client. The function of TC1 is to act as a proxy and to intercept the communications of "Component 1" (C1) providing an archive of the communication history during the states synchronization stage. Important to realize that C1 is, in the considered use case, a composite component, i.e., can be a control chart that contains function blocks. The component is thus iterated to locate all the communication components, which are either a "setVar" that is used to write variables, or a "getVar" that reads variables, at the server side. Once the components are located, all the writing operations are rerouted to the traffic controller path instead of the server. An identical copy of the write operation and its time stamp are made and pushed into the PB. On the other hand, all the located read operations are transformed into read requests that are forwarded to TF1. In a similar manner, the read operations are executed by TF1 and the returned values are forwarded to the C1 and a copy with the time stamps are saved in the PB.

Component Migration

The component is migrated using JSON format as described in the implementation (cf. Sec. 9.1.13). Upon successful component migration, Traffic Controller 2 (TC2) is created.

Synchronization

Initially, once a write operation from "Component 1"" (C1") is executed, TF2 searches in the PB for a similar entry. Once found, the replay phase initiates. The replay phase takes

place between C1" and TC2 dequeuing the entries in the PB. An empty PB indicates a synchronized state between C1 and C1". In case the migration here performs a redundancy component creation operation, the process terminates here.

Handover

This stage exists only if it is a load balancing operation. A final check between TC1 and TC2 is done to ensure similar writing operation from C1 and C1". Upon a successful return, the handover phase is executed terminating the communication between the server and C1 and setting a direct communication between C1" and the server without traffic controllers. The traffic controllers and C1 are then deleted. Fig. 11.2 shows the sequence diagram of a successful component redeployment.

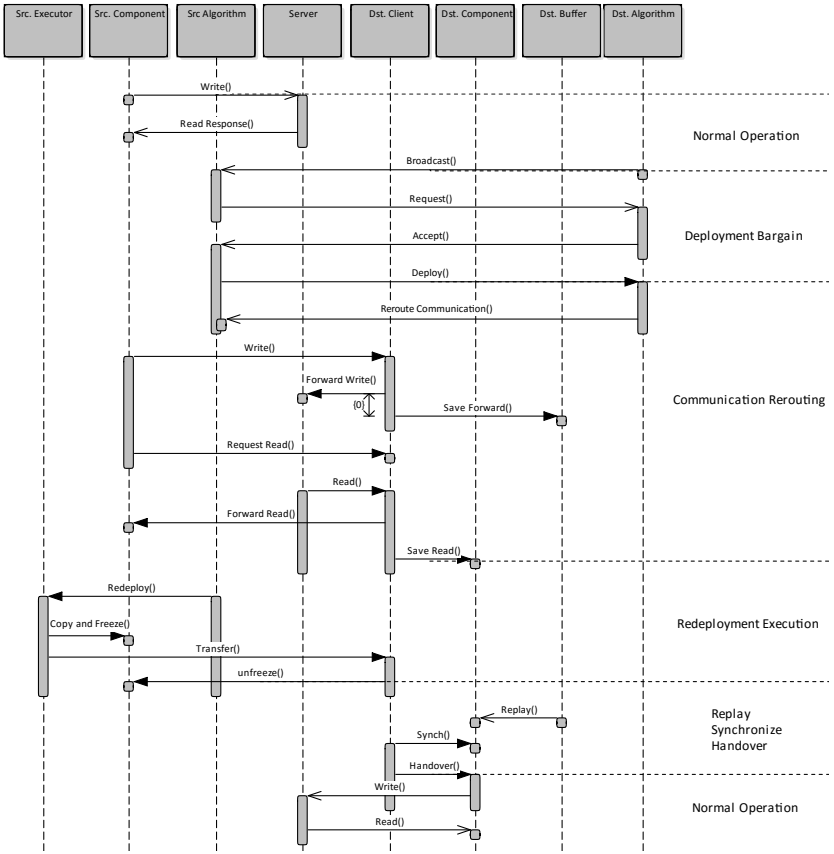


Figure 11.2: Sequence Diagram of a Successful Component Redeployment between two Nodes

11.1.3 Improvements for Load Model

The presented load model can be improved by considering the following points:

- The temporary load (processing and communication load) induced by a load transfer is considered negligible in the presented model which can be taken into consideration for a more realistic modeling.
- The considered load is assumed to be static and does not vary throughout time. A dynamic load can be considered instead of a static load. The load should depend on the component overhead variations, e.g., best and worst case processing overhead.

11.1.4 Improvements in the Decentralized Algorithm

The BRAD algorithm presented in Sec. 8.1.3 can be enhanced via the following:

Bottlenecks

Network topologies that enclose bottlenecks can impair the algorithm performance. A bottleneck exists for example in $K_n - K_n$ topologies as shown in Fig. 11.3. The bottlenecks can cause load congestions as the load cannot traverse smoothly between the network ends. Identifying bottlenecks, by recognizing the deviations of the node valencies, can be useful to improve the performance. Once a bottleneck is detected, special conditions should be applied for the bottleneck nodes in terms of load carrying. A bottleneck node should act as a bypassing bridge, i.e., does not keep loads rather only allows the load flow. Once the saturation phase is reached, this condition can be rectified so that the bottleneck nodes contribute to the load balancing.

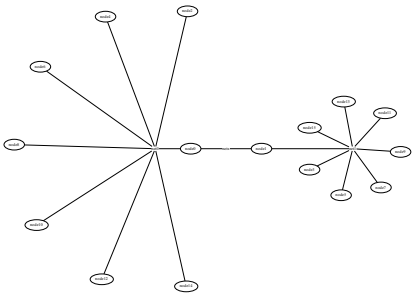


Figure 11.3: Illustration of a Bottleneck in a 16 Nodes Kn-Kn Networks

Simulated Annealing Mechanism

The presented algorithm offers a heuristic solution and does not guarantee providing a global optimum solution. In some cases, the algorithm can stumble upon a solution and gets stuck at a local optimum. Similar to the simulated annealing probabilistic technique, after reaching saturation and the load balancing converges, the algorithm can intentionally

perform load transfers that does not better (decrease) the global TSE of the network. A load transfer can either worsen (increase) or not affect (maintains the same) global TSE. This technique can help the algorithm escape a local optimum in order to find a global optimum. The simulated annealing inspired excitation can be also performed in a simulation mode, i.e., no real load transfers are performed rather only a simulated version to ensure that the eventual convergence reaches a better state before starting. The excitation can be performed on two stages starting by performing non affecting (to the TSE) load transfers and followed by the second stage where the load transfers can worsen the TSE.

Optimizing the Optimization

Similarly, the whole optimization can be executed in a simulation mode. This enables an optimization of the optimization. In other words, the final load profile can be retrieved and hence a further optimization can be executed to compute the best way, i.e., least number of load transfers, to reach the final profile. Furthermore, the time step of the BRAD state machine can be optimized. In the current implementation, the time step between the different states is fixed to the maximum time duration of moving the largest application in the network. Foreseeing the load transfers can be used to minimize and dynamically tune the time step parameter to enhance the performance.

Further Optimization Criteria

Other optimization criteria (cf. Sec 3.4.1) can be considered as objectives for the algorithm, e.g., redundancy. Initially, the degree of redundancy can be considered by replicating the software components in the network. Each redundancy component shall be given a unique identification which is derived from the original component's identification. During the load balancing stage, a constraints is enforced such that redundant components are never moved to a node where one of the copies or the original component exists. Distributing the redundant copies on different nodes ensures the intended objective of securing the application from node outages.

11.2 Improvements in the Infrastructure

During redeployment, specifically in the synchronization phase, the writing and reading requests are performed via an intermediate component as described in Sec. 11.1.2. The QoS of the communication is not taken into consideration, which can cause jitter in the performance of the original source component control execution. Implementing a network component that establishes communication contracts with QoS with other computation nodes in the network can provide improvements to the performance problem.

11.3 Further Utilizations of Agents Systems Approach

In the agents systems approach, a brief overview over the concept and how load balancing can be performed was given. Further points regarding relevant data distribution in the cloud (e.g., asset administration shell) and how load balancing can improve the QoS of the data retrieving should be considered.

Bibliography

[1] I40 Glossar, URL: /www.plattform-i40.de/i40/navigation/de/service/glossar/glossar.

[2] MIT online lecture notes, URL: http://web.mit.edu/2.14/www/psets/nov17.pdf.

[3] Platform Industrie 4.0. Aspekte der Forschungsroadmap in den Anwendungsszenarien, 2013.

[4] Niels Henrik Abel. *Mémoire Sur Les Équations Algeébriques Où on Démontre L'impssibilité de la Résolution de L'equation Générale Du Cinquième Dégré... Christiania, Groendahl, 1824*. University of Olso, 1824.

[5] Thomas Aicher, Markus Spindler, Johannes Fottner, and Birgit Vogel-Heuser. Analyzing the industrial scalability of backwards compatible intralogistics systems. *Production Engineering*, 12(3-4):297–307, 2018.

[6] Raymond G Ayoub. Paolo ruffini's contributions to the quintic. *Archive for history of exact sciences*, 23(3):253–277, 1980.

[7] Mahyar Azarmipour, Julian Alexander Grothoff, Haitham Ahmed Elfahaam, and Ulrich Epple. Hypervisor-basierte Virtualisierung in der industriellen Automation. In *[19. Leitkongress der Mess- und Automatisierungstechnik, 2018-07-03 - 2018-07-04, Baden-Baden, Germany]*, pages 467–480. 19. Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden (Germany), 3 Jul 2018 - 4 Jul 2018, Jul 2018.

[8] C Bert. An improved approximation for settling time of second-order linear systems. *IEEE transactions on automatic control*, 31(7):642–643, 1986.

[9] Theresa Beyer, Peter Göhner, Ramin Yousefifar, and Karl-Heinz Wehking. Agent-based dimensioning to support the planning of intra-logistics systems. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE, 2016.

[10] Theresa Beyer, Nasser Jazdi, Peter Göhner, and Ramin Yousefifar. Knowledge-based planning and adaptation of industrial automation systems. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–4. IEEE, 2015.

[11] Theresa Beyer, Ramin Yousefifar, Sebastian Abele, Manuel Bordasch, Peter Göhner, and Karl-Heinz Wehking. Flexible agent-based planning and adaptation of material handling systems. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1060–1065. IEEE, 2015.

[12] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*, volume 290. Citeseer, 1976.

[13] Stephen Boyd. Convex optimization of graph laplacian eigenvalues. In *Proceedings of the International Congress of Mathematicians*, volume 3, pages 1311–1319, 2006.

[14] Stephen Boyd, Persi Diaconis, Pablo Parrilo, and Lin Xiao. Fastest mixing markov chain on graphs with symmetries. *SIAM Journal on Optimization*, 20(2):792–819, 2009.

[15] Stephen Boyd, Persi Diaconis, Pablo Parrilo, and Lin Xiao. Fastest mixing markov chain on graphs with symmetries. *SIAM Journal on Optimization*, 20(2):792–819, 2009.

[16] Junwei Cao. Self-organizing agents for grid load balancing. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 388–395. IEEE Computer Society, 2004.

[17] S. Cheshire and M. Krochmal. Dns-based service discovery.

[18] Stuart Cheshire and Marc Krochmal. Rfc 6762: Multicast dns. *Internet Engineering Task Force (IETF) standard*, 2013.

[19] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[20] International Electrotechnical Commission et al. Iec 61512 batch control. parts 1-4, 2000.

[21] Giuseppe Confessore, Stefano Giordani, and Silvia Rismondo. A market-based multi-agent system model for decentralized multi-project scheduling. *Annals of Operations Research*, 150(1):115–135, 2007.

[22] George Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of parallel and distributed computing*, 7(2):279–301, 1989.

[23] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[24] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[25] Docker. Documentation of docker, Nov. 2018.

[26] Karin Eckert, Alexander Fay, Thomas Hadlich, Christian Diedrich, Timo Frank, and Birgit Vogel-Heuser. Design patterns for distributed automation systems with consideration of non-functional requirements. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, pages 1–9. IEEE, 2012.

[27] Sherihan Abu Elenin and Masato Kitakami. Comparing static load balancing algorithms in grid. In *International Conference on Cooperative Design, Visualization and Engineering*, pages 170–177. Springer, 2011.

[28] Sherihan Abu Elenin and Masato Kitakami. Performance analysis of static load balancing in grid. *International Journal of Electrical & Computer Sciences IJECS-IJENS*, 11(3), 2011.

[29] Haitham Ahmed Elfahaam, Florian Palm, Constantin August Wagner, Mahyar Azarmipour, and Ulrich Epple. Redeployment in Industrie 4.0. In *[19. Leitkongress der Mess- und Automatisierungstechnik, 2018-07-03 - 2018-07-04, Baden-Baden, Germany]*. 19. Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden (Germany), 3 Jul 2018 - 4 Jul 2018, Jul 2018.

[30] Haitham Elfaham, Constantin Wagner, Sten Grüner, Lars Nothdurft, and Ulrich Epple. A modular benchmark for evaluating load distribution algorithms. In *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*, pages 4863–4870. IEEE, 2016.

[31] Ulrich Epple. Konzepte-agentensysteme in der leittechnik-zur erfullung zukunftiger anforderungen muss das leittechnische softwarekonzept innoviert werden. agentensysteme konnten eine losung sein. *Automatisierungstechnische Praxis*, 42(8):45–51, 2000.

[32] Ulrich Epple. Prozessführung als systemfunktion. *Integration von Advanced Control in der Prozessindustrie: Rapid Control Prototyping*, pages 173–200, 2008.

[33] Ulrich Epple. Agentenmodelle in der anlagenautomation. In *Agentensysteme in der Automatisierungstechnik*, pages 95–110. Springer, 2013.

[34] Alexander Fay, Birgit Vogel-Heuser, Timo Frank, Karin Eckert, Thomas Hadlich, and Christian Diedrich. Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns. *Journal of Systems and Software*, 101:221–235, 2015.

[35] Juliane Fischer, Marga Marcos, and Birgit Vogel-Heuser. Model-based development of a multi-agent system for controlling material flow systems. *at-Automatisierungstechnik*, 66(5):438–448, 2018.

[36] Forschungsunion and acatech. Recommendations for implementing the strategic initiative INDUSTRIE 4.0, 2013.

[37] Kunjal Garala, Namrata Goswami, and Prashant D Maheta. A performance analysis of load balancing algorithms in cloud environment. In *2015 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–6, 2015.

[38] Felix Gehlhoff, Tobias Linnenberg, and Alexander Fay. Optimierung von auktionsmechanismen. *atp magazin*, 59(09):54–66, 2017.

[39] Peter Göhner. *Agentensysteme in der Automatisierungstechnik*. Springer-Verlag, 2013.

[40] Thomas Goldschmidt and Stefan Hauck-Stattelmann. Software containers for industrial control. In *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on*, pages 258–265. IEEE, 2016.

[41] Farid Golnaraghi and BC Kuo. Automatic control systems. *Complex Variables*, 2:1–1, 2010.

[42] Keerthana Govindaraj and Alexander Artemenko. Container live migration for latency critical industrial applications on edge computing. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 83–90. IEEE, 2018.

[43] Julian Alexander Grothoff, Constantin August Wagner, and Ulrich Epple. BaSys 4.0: Metamodell der Komponenten und Ihres Aufbaus; 1st ed. Technical Report D-PC2.4, Aachen, 2018. Veröffentlicht auf dem Publikationsserver der RWTH Aachen University.

[44] Sten Grüner. *Ressourcenadaptive Anwendungen für die operative Prozessleittechnik*. VDI Verlag GmbH, 2017.

[45] Sten Grüner, Somayeh Malakuti, Johannes Schmitt, Tarik Terzimehic, Monika Wenger, and Haitham Elfaham. Alternatives for flexible deployment architectures in industrial automation systems. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 35–42. IEEE, 2018.

[46] Hans-Ulrich Heiss and Achim Payer. Paste: A tool for evaluation of processor allocation strategies. In *Proc. 6th Int. Conf. on Modelling Tools and Techn. for Comp Perf. Eval*, pages 367–371, 1992.

[47] Hans-Ulrich Heiss and Michael Schmitz. Decentralized dynamic load balancing: The particles approach. *Information Sciences*, 84(1):115–128, 1995.

[48] Michael R Hines, Umesh Deshpande, and Kartik Gopalan. Post-copy live migration of virtual machines. *ACM SIGOPS operating systems review*, 43(3):14–26, 2009.

[49] Roger A Horn. Cr johnson matrix analysis, 1985.

[50] Sergii Iarovyi, Wael M Mohammed, Andrei Lobov, Borja Ramis Ferrer, and Jose L Martinez Lastra. Cyber–physical systems for open-knowledge-driven manufacturing execution systems. *Proceedings of the IEEE*, 104(5):1142–1154, 2016.

[51] Apple Inc. White paper: Bonjour technology overview. Technical report, EECS at UC Berklery, 2002.

[52] American National Standards Institute. *ANSI-ISA-88.00. 01-2010: Batch Control Part 1: Models and Terminology*. ISA, 2010.

[53] YOSHIHIRO Kanno, Kazuo Murota, and Naoki. Katoh. Group symmetry in interior-point methods for semidefinite program. *Optimization and Engineering*, (2):293–320, 2001.

[54] Mayanka Katyal and Atul Mishra. A comparative study of load balancing algorithms in cloud computing environment. *arXiv preprint arXiv:1403.6918*, 2014.

[55] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[56] Michal Kočvara and Michael Stingl. Pennon: A code for convex nonlinear and semidefinite programming. *Optimization methods and software*, 18(3):317–333, 2003.

[57] Zhou Lei, Exiong Sun, Shengbo Chen, Jiang Wu, and Wenfeng Shen. A novel hybrid-copy algorithm for live migration of virtual machine. *Future Internet*, 9(3):37, 2017.

[58] Tobias Linnenberg, Ireneus Wior, and Alexander Fay. Analysis of potential instabilities in agent-based smart grid control systems. In *IECON 2013-39th Annual Conference of the IEEE Industrial Electronics Society*, pages 7422–7427. IEEE, 2013.

[59] Tobias Linnenberg, Ireneus Wior, Sebastian Schreiber, and Alexander Fay. A market-based multi-agent-system for decentralized power and grid control. In *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1–8. IEEE, 2011.

[60] Johan Lofberg. Yalmip: A toolbox for modeling and optimization in matlab. In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 284–289. IEEE, 2004.

[61] Michele Lombardi and Michela Milano. Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints*, 17(1):51–85, 2012.

[62] Simone A Ludwig and Azin Moallem. Swarm intelligence approaches for grid load balancing. *Journal of Grid Computing*, 9(3):279–301, 2011.

[63] Hamid Mcheick, Ziad Rajih Mohammed, and Abbass Lakiss. Evaluation of load balance algorithms. In *Software Engineering Research, Management and Applications (SERA), 2011 9th International Conference on*, pages 104–109. IEEE, 2011.

[64] Alberto Montresor, Hein Meling, and Özalp Babaoğlu. Messor: Load-balancing through a swarm of autonomous agents. In *Agents and Peer-to-Peer Computing*, pages 125–137. Springer, 2002.

[65] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[66] Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. SIAM, 1994.

[67] Reza Olfati-Saber, J Alex Fax, and Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

[68] Diego Ongaro and John K Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, pages 305–319, 2014.

[69] Wei Peng, Hong Li, Min Yao, and Zheng Sun. Deployment optimization for autosar system configuration. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, volume 4, pages V4–189. IEEE, 2010.

[70] Martin Polke. *Process control engineering*. John Wiley & Sons, 2008.

[71] Florian A. Potra and Stephen J. Wright. Interior-point methods. *J. Comput. Appl. Math.*, 124(12):281–302, 2000.

[72] Abhijit A Rajguru and SS Apte. A comparative performance analysis of load balancing algorithms in distributed system using qualitative parameters. *International Journal of Recent Technology and Engineering*, 1(3):175–179, 2012.

[73] ITUTX Recommendation. 902 (1995)— iso/iec 10746-2: 1996. *Information technology–Open Distributed Processing–Reference Model: Foundations*.

[74] Daniel Regulin, Amelia Glaese, Stefan Feldmann, Daniel Schütz, and Birgit Vogel-Heuser. Enabling flexible automation system hardware: Dynamic reconfiguration of a real-time capable field-bus. In *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 1198–1205. IEEE, 2015.

[75] Daniel Regulin, Daniel Schütz, Thomas Aicher, and Birgit Vogel-Heuser. Model based design of knowledge bases in multi agent systems for enabling automatic reconfiguration capabilities of material flow modules. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 133–140. IEEE, 2016.

[76] Sebastian Rehberger, Lucas Spreiter, and Birgit Vogel-Heuser. An agent-based approach for dependable planning of production sequences in automated production systems. *at-Automatisierungstechnik*, 65(11):766–778, 2017.

[77] Rohit Saxena, Ankur Kumar, Anuj Kumar, and Shailesh Saxena. Distributed and grid computing: An analytical comparison.

[78] Shailesh Saxena, Mohd Zubair Khan, and Ravendra Singh. Performance analysis in distributed system of dynamic load balancing using fuzzy logic. In *Engineering and Technology (S-CET), 2012 Spring Congress on*, pages 1–5. IEEE, 2012.

[79] Günther Schuh, Reiner Anderl, Jürgen Gausemeier, Michael ten Hompel, and Wolfgang Wahlster. Industrie 4.0 maturity index. *Managing the Digital Transformation of Companies. Munich: Herbert Utz*, 2017.

[80] Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma. Performance analysis of load balancing algorithms. *World Academy of Science, Engineering and Technology*, 38(3):269–272, 2008.

[81] Roopak Sinha, Kenneth Johnson, and Radu Calinescu. A scalable approach for reconfiguring evolving industrial control systems. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pages 1–8. IEEE, 2014.

[82] Dieter Steegmüller and Michael Zürn. Wandlungsfähige produktionssysteme für den automobilbau der zukunft. In *Industrie 4.0 in Produktion, Automatisierung und Logistik*, pages 103–119. Springer, 2014.

[83] Michael Steiger. Fault-tolerant turbine controller. *Master's thesis*, 2008.

[84] Docker Swarm. Accessed on jan. 2017.

[85] Tarik Terzimehic, Sebastian Voss, and Monika Wenger. Using design space exploration to calculate deployment configurations of iec 61499-based systems.

[86] Marvin M Theimer, Keith A Lantz, and David R Cheriton. *Preemptable remote execution facilities for the V-system*, volume 19. ACM, 1985.

[87] Michael Tiegelkamp and Karl-Heinz John. *IEC 61131-3: Programming industrial automation systems*. Springer, 1995.

[88] Reha H Tütüncü, Kim-Chuan Toh, and Michael J Todd. Solving semidefinite-quadratic-linear programs using sdpt3. *Mathematical programming*, 95(2):189–217, 2003.

[89] Richard S Varga. *Matrix iterative analysis*, volume 27. Springer Science & Business Media, 2009.

[90] Richard S Varga. *Matrix iterative analysis*, volume 27. Springer Science & Business Media, 2009.

[91] Valeriy Vyatkin and Instrument Society of America. *IEC 61499 function blocks for embedded and distributed control systems design*. ISA-Instrumentation, Systems, and Automation Society Oneida, 2007.

[92] Constantin Wagner. *Ein Konzept zur Unterstützung der Wiederverwendung in komponentenbasierten verteilten Systemen der operativen leittechnik*. VDI Verlag GmbH, 2018.

[93] Constantin Wagner, Julian Grothoff, Ulrich Epple, Rainer Drath, Somayeh Malakuti, Sten Grüner, Michael Hoffmeister, and Patrick Zimermann. The role of the industry 4.0 asset administration shell and the digital twin during the life cycle of a plant. In *Emerging Technologies and Factory Automation (ETFA), 2017 22nd IEEE International Conference on*, pages 1–8. IEEE, 2017.

[94] Constantin Wagner, David Kampert, Andreas Schüller, Florian Palm, Sten Grüner, and Ulrich Epple. Model based synthesis of automation functionality. *at-Automatisierungstechnik*, 64(3):168–185, 2016.

[95] Michael Wahler and Manuel Oriol. Disruption-free software updates in automation systems. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pages 1–8. IEEE, 2014.

[96] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[97] Liyong Yu, Sten Grüner, and Ulrich Epple. An engineerable procedure description method for industrial automation. In *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*, pages 1–8. IEEE, 2013.

Die Reihen der Fortschritt-Berichte VDI:

1 Konstruktionstechnik/Maschinenelemente
2 Fertigungstechnik
3 Verfahrenstechnik
4 Bauingenieurwesen
5 Grund- und Werkstoffe/Kunststoffe
6 Energietechnik
7 Strömungstechnik
8 Mess-, Steuerungs- und Regelungstechnik
9 Elektronik/Mikro- und Nanotechnik
10 Informatik/Kommunikation
11 Schwingungstechnik
12 Verkehrstechnik/Fahrzeugtechnik
13 Fördertechnik/Logistik
14 Landtechnik/Lebensmitteltechnik
15 Umwelttechnik
16 Technik und Wirtschaft
17 Biotechnik/Medizintechnik
18 Mechanik/Bruchmechanik
19 Wärmetechnik/Kältetechnik
20 Rechnerunterstützte Verfahren (CAD, CAM, CAE CAQ, CIM ...)
21 Elektrotechnik
22 Mensch-Maschine-Systeme
23 Technische Gebäudeausrüstung

ISBN 978-3-18-526708-6