Managing complexity of Digital Twin models

Development of NewMODE as a network theory approach to model decomposition

Laura Crowley | 4827732 | Master Thesis | August 2020

TNO innovation for life

TU Delft

MANAGING COMPLEXITY OF DIGITAL TWIN MODELS

Development of NewMODE as a network theory approach to model decomposition

Master thesis submitted to Delft University of Technology
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE
in **Engineering and Policy Analysis**
Faculty of Technology, Policy, and Management

by
Laura Crowley
Student number: 4827732

To be defended in public on 10th August 2020

Graduation Committee

Chairperson:	Dr. Martijn Warnier	Systems Engineering and Simulation, TU Delft
First Supervisor:	Dr. Yilin Huang	Systems Engineering and Simulation, TU Delft
Second Supervisor:	Dr.ir Jan Kwakkel	Policy Analysis, TU Delft
External Supervisors:	Elena Lazovik	Monitoring and Control Services, TNO
	Dr. Paolo Pileggi	Monitoring and Control Services, TNO
	Dr. Jacques Verriet	Embedded Systems Innovation, TNO

TU Delft Delft University of Technology TNO innovation for life ESI

Summary

Modularity is desirable in many domains; from shaping organizational structures, product design, to software and engineering it is coveted as a means to manage system complexity [1]. In general, modularity is a system's property that describes the extent to which it can be separated and recombined [2]. In software systems, modularity is often sought in the form of Service Oriented Architecture (SOA) and, more recently, microservices architecture [3]. Specifically in Modelling & Simulation, modularity is often synonymous with component based modelling. In both cases the goal is clear; to identify components that are highly cohesive and loosely coupled with a well-defined interface [4]. A point less argued in literature is that identifying components within models that are fit for modularity is a tedious effort that encompasses many dimensions of modelling and software knowledge and often relies heavily on the experience of experts.

This thesis aims to build on model decomposition know-how and heuristics from research and industry, and formalize it with the goal of building an automated decomposition methodology that enables engineers to identify candidate decomposition architectures. In general, the most common approach to model decomposition involves ad-hoc design choices based on the original modeller's experience and intuition. Although this is successful on a case-by-case basis it is time-consuming and depends on the availability of expert knowledge as well as an understanding of the legacy system. It also challenges the reproducibility of the decomposition. If, instead, we consider model decomposition as an act of analysing complexity itself it is possible to formalize an approach to model decomposition which leads to knowledge transfer and automation of this process.

NewMODE is proposed as a novel method for model decomposition that is analytical, automated and reproducible. By leaning on ideas from Model Driven Reverse Engineering (MDRE) and Model Driven Development (MDD) a metamodel is used to define the original model to be decomposed in the form of a three-layered, directed network. Figure 1 shows an example of such a metamodel in (a). By applying a series of horizontal transformations defined on this metamodel [5], candidate decompositions are produced as (b). The proposed decomposition algorithm results in a number of components (c), as well as the set of boundary edges (d) that were removed to isolate these components. The transformation from (d) to (e) completes the decomposition by removing these boundary edges such that inter-component synchronisation is correctly maintained.

NewMODE is applied iteratively to find a set of candidate decomposition architectures of different levels of granularity, where granularity refers to the number of components in the system [6]. NewMODE is evaluated for a class of models in the logistics and process modelling application of Digital Twin. Specifically, the method is implemented for models built with the Logistics Specification and Analysis Tool (LSAT). By comparing the results of the proposed decomposition method to that of a manual decomposition by an expert, it is found that the method generated candidate solutions that achieved lower internal complexity and smaller external interfaces than the manual decomposition.

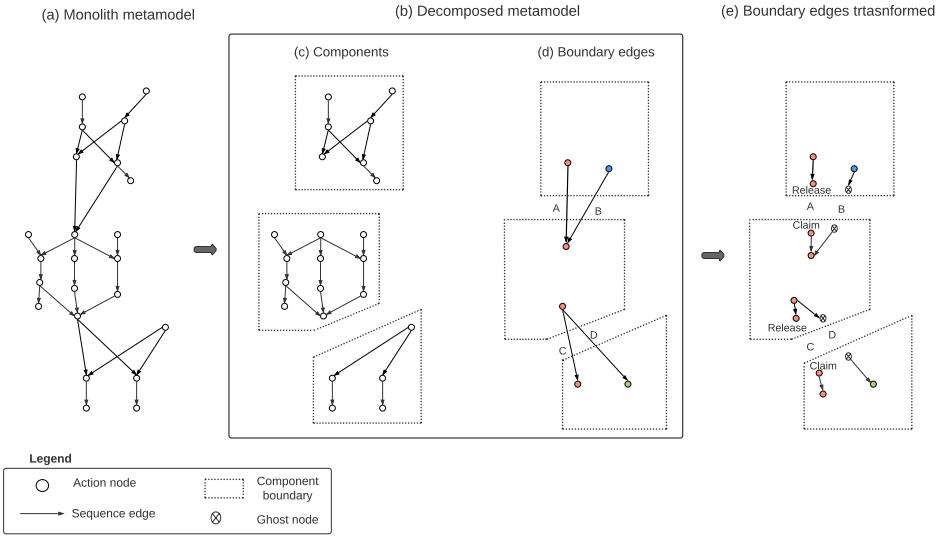


Figure 1: Overview of the decomposition algorithm. From left to right: the metamodel of the monolith (a), is decomposed into separate components (C), and the set of boundary edges that isolate these components (d). The final transformation deals with inter-component synchronisation (e). In (d) and (e) the colour of the action node represents the resource on which that action is executed.

Contents

List of Figures

List of Tables

1

Introduction to model decomposition

1.1 Complexity of Digital Twin models

The synergy of emerging technologies is leading to digitalisation of all domains in industry as many organisations are turning to virtual representations of their assets in order to manage physical systems. In 2002 the term Digital Twin was first used to describe this digital information as a 'twin' of the information that is embedded within the physical system throughout its lifecycle [8]. Before this term was coined, this has been referred to as a computational megamodel, a synchronised virtual prototype, or a device shadow but can be defined loosely as any adaptive model of a complex physical system [9].

The adoption of Digital Twin in industry is particularly evident in manufacturing domains, overlapping with the concept of Industry 4.0 where the aim is to build smart factories and cyber-physical systems. The possibility of real-time autonomous control and optimization of industrial processes is especially attractive considering the global demand for high-quality and short-delivery products. At the same time, industrial processes controlled by a Digital Twin benefit from increased efficiency, cost-reduction, higher product quality and reducing harmful emissions. For example, many of the leading producers in the industry of semi-conductor chips rely on detailed models of their complex industrial processes to design and monitor production. In the design phase this means performing throughput analysis and scenario discovery. In the operational phase, predictive maintenance can be performed by comparing the output data collected from the physical system with the expected calculations from the Digital Twin model.

Simulation is at the heart of Digital Twin; it provides the technology to fuse physics-based and data-driven modelling which mirrors the fusion of real and virtual worlds [10]. Although modelling and simulation has been used in many ways in the last century its most recent and widespread adoption in industry is almost synonymous with the technology of Digital Twin. In fact, the role of simulation is considered intrinsic to advancements in digitalisation as it provides a platform to synergise technologies such as high performance computing, IoT, 5G technology, and Virtual Reality [11].

As an emerging technology, the challenges to realising Digital Twin are vast and tackling these challenges is resulting in an overhaul of traditional modelling methodology to allow for increased flexibility and autonomy in systems. Models used in industry are becoming more detailed and interconnected. Traditional models that are built with a monolith architecture cannot deal with this increasing complexity. Recently, the Functional Mockup Interface has been proposed as an interface for implementing modular architectures in Digital Twin models [12]. In fact the literature shows a general consensus that distributed, modular simulation architectures are necessary to operationalise Digital Twin, although the details of these implementations remain under discussion in literature. Within a single organisation, a modular system increases the maintainability of code and the reusability of existing model functionality. However, adopting this approach requires configuring the communication between components which leads to difficult challenges surrounding network latency, interoperability, real-time data assimilation and synchronisation of simulation logic (co-simulation).

Digital Twin is a revolutionary technology that cuts across many domains in industry. The underlying simulation models in a Digital Twin are themselves growing in complexity, which mirrors the complexity of the physical process they represent [13]. New techniques in managing complexity of models are needed to operationalise Digital Twin.

1.2 Modularity in systems design

The degree to which a system can be separated and subsequently recombined is referred to as the modularity of the system [2]. Modularity owes its popularity to the fact that its benefits are desirable across many domains; from shaping organizational structure, product design, to software and engineering it is often desired as a means to manage the complexity of a system. Many academic disciplines have been concerned with modularity over the past few decades, and it forms the core in the academic field of systems science where Simon's parable of the two watchmaker is considered one of the first descriptions of a modular system [14]. In the field of modelling and simulation (M&S), modularity is often synonymous with component-based modelling whereby components have a clear functionality, the desired level of granularity, well-defined interfaces and are packaged neatly with their dependencies. Understandably, this has many attractions to modellers: reusability, flexibility, dependability and maintainability to name a few [15]. Today, challenges of how to realise a modular design in modelling remains elusive. Standard practice is to build complex models from scratch and implement simulation logic for each new project. When it comes to reuse of components, this is typically done by the person who wrote it themselves, if at all.

In the field of software engineering (SWEng), complexity is a commonly cited challenge and modular solutions are often sought in the form of Service Oriented Architecture (SOA) or microservices [3]. The complexity in SWEng is a result of increasingly large web-based applications that are run in a monolithic environment. Increasing the size of development teams proportionally to the size of the system does not stem the delays in sprint deployment. In 2011, Netflix developed their platform in a microservices architecture in order to increase maintainability of the system [16]. Components are developed, tested, maintained and redeployed by a small team without necessitating redesign of the entire system logic. This formed the first implementation of a large microservices architecture which is not widely

considered the golden ticket to achieving modular design in SWEng. By decomposing the model into a number of smaller, atomic, loosely-coupled services they can then be distributed on a set of virtual machines within the cloud, or physical machines in different locations [17]. These resultant individual components are referred to as the microservices. The necessary performance for real-time use of Digital Twin models can be achieved in a cloud native, microservices architecture [18], but migrating to this system design is not straightforward.

Most patterns for modular system design, pre-suppose the existence of model components, or a readily decomposable model code. Writing these components from scratch would be undesirable for companies who already have legacy models. Although they may not be in the desired architectural format, existing monoliths still hold embedded functionality that can be re-used if extracted as components. Even when building new systems, it is often easier to build a model in a monolithic way and only as it grows in complexity do developers decompose the model. This approach, described by Newman as the monolith-first technique, therefore necessitates *monolith decomposition* [19].

The task of model decomposition is labour intensive and poorly documented in literature. It relies on an expert understanding of the legacy system and ad-hoc design choices which can lead to project delays and discrepancies in behaviour after migration. The original developer of the monolith may have left the team and documentation is typically incomplete and out-dated. Additionally, manual decomposition does not guarantee that the decomposed system is optimal for the new architecture.

Overall, modular systems provide an opportunity to manage complexity in simulation models but adopting a modular approach is daunting for any organisation. An important question in modularity is how to split a complex system into a set of components. Model decomposition has the potential to breathe life into legacy models by extracting embedded functionality as components that can be re-used. Furthermore, model decomposition is essential when developing Digital Twin models in a monolith-first approach. Therefore, there is a clear motivation from industry and academic trends to formalise model decomposition. This thesis aims to improve on know-how and heuristics from industry in combination with theoretical foundations from multiple academic domains. The goal is to build an automated decomposition methodology that enables engineers to manage complexity in Digital Twin simulation models.

1.3 Challenges in model decomposition

Decomposition involves breaking up a system into separate components. This is a trivial task when a system naturally consists of isolated, independent parts whose internal behaviour is stronger than the interaction between them. However there are many models for which this is not the case. In fact, the motivation for modelling a system is that the interaction between parts of the system are themselves not-straightforward and so the system is studied as a whole [14]. Considering a model of such a system, decomposition becomes challenging. It may even seem counter-intuitive to decompose a system that is being studied for its complexity in the first place. However, it is necessary if people are to be able to comprehend the model, analyse it, maintain the code base, or run it using microservices technology.

There is convergence across domains in literature that a good decomposition achieves

high cohesion and loose coupling but the terms are still abstract concepts and provide little guidance in how to achieve these goals. The degree of coupling refers to the interaction between components in the form of data transfer, communication, or modelling assumptions [4]. Cohesion can be described as the degree to which a component serves a single, united purpose. There is equally sparse instruction of how to evaluate the degree of coupling between components or the degree of cohesion in a component. One can imagine that being able to quantify a decomposition based on criteria has benefits for reproducibility as well as reducing the time and expertise required to extract a good decomposition from a model.

Another concept that is often referred to is the level of granularity in a decomposition. This refers to balancing the size and number of components in the decomposed system [6]. Intuitively, one could achieve perfectly high cohesion with complete granularity by breaking the system into its smallest units. Of course, this is likely to have negative effects on the coupling. This is the intuition behind the granularity trade-off. However, considering complex systems that are decomposed into heterogeneous parts we do not expect uniformly sized components. This suggests that the granularity trade-off is not directly linear and therefore it is hard to select the number of components in a decomposition.

In general, the most common approach to model decomposition involves ad-hoc design choices based on the original modeller's experience and intuition. Although this is successful on a case-by-case basis it is time-consuming and depends on the availability of expert knowledge as well as an understanding of the legacy system. It also challenges the reproducibility of the decomposition. If, instead, we consider model decomposition as an act of analysing complexity itself it is possible to formalize an approach to model decomposition which leads to knowledge transfer and automation of this process.

Model decomposition requires reasoning of the model on the architectural level with the ability to move fluidly between the model code and its structure. First, we must consider a model as some code written in a specific modelling formalism [20]. If we strive to define decomposition criteria that is language agnostic, reasoning of decomposition cannot be done on the code level. This is not only motivated by a desire to be as general as possible but by the difficulty it would bring to reason about component architecture on the code level [21]. This is because models have been built for a specific purpose and this purpose is not to aid decisions of its own composition. Therefore, we prefer a metamodel that can describe the model's architecture. The architecture of a model is an abstract concept, analogous to the architecture of a building [22]. It functions as a blueprint for the model, explicitly laying out the relations and boundaries within the model in a way that facilitates higher reasoning of its design. This architectural representation is known as a metamodel, specifically a *token* model [23] of a model.

1.4 Research gap

When we speak of the 'Art and science of modelling' [24] it allows us to place model decomposition into the category of art where the methodology is elusive, the reasoning is undetermined and success is dependent on the expertise of the modeller. It is a process that is developed in our thoughts and minds and sometimes difficult to articulate. Therefore it is difficult to transfer the knowledge of model decomposition and so it remains elusive. The gap in literature for consideration of model decomposition as a form of scientific analysis

is wide open and forms the main objective of this thesis.

This gap is a valuable direction for research, due to the proof of analytic approaches to system decomposition in domains that are not specific to modelling. Specifically, in software engineering there is an increasing number of studies in automated re-architecting of legacy and operational software systems. This is discussed in detail in Section 2.3 and the main conclusion is that analytic approaches to software re-engineering provide an outline of potential approaches to model decomposition, although the details of these implementations require significant adaption.

The trend in the literature towards automation has practical benefits for time and expertise required for model decomposition. Equally this contributes to the consistent challenge in academic simulation studies of reproducibility. By reducing the dependency on the developer's design choices for model decomposition, the results of component based model building are more reproducible. This is another justification for tacking the academic gap of formal decomposition techniques. Most studies accept the component architecture as a design choice that relies on experience. In fact, formalising a method for model decomposition is highly novel and addresses a clear gap in literature and industry.

1.5 Research questions

Based on the research gap, the goal of this research is:

Develop an analytical approach to model decomposition that is semi-automated and reproducible.

Designing this novel approach leads to three research questions that demarcate the main challenges in model decomposition of a legacy system. Monolith models are either the result of a legacy system that has evolved over time with poor design documentation, or a monolith-first approach to model building. In both cases, albeit a larger challenge in the former, it is impossible to decompose the model without some representation of its architecture [25]. Simply, it is impractical to reason on the architecture of a model at the code level, but by transforming the monolith model code to a representation that facilitates understanding, decomposition is possible. Therefore, this is the focus of the first research question:

RQ1: How can the architecture of a complex model be represented?

Given a representation of the model, it is necessary to get to the core of this research; apply some technique that can identify components within a model. There are guiding principles in design choices for model decomposition, although they are rarely formalised in literature [26]. The challenge is identifying strategies that are applicable for a formal or automated approach, and adapting them accordingly. Therefore the second research question is:

RQ2: How can the architectural metamodel be decomposed?

A challenge in model decomposition is the inherent trade-off between desired characteristics as the model is decomposed. The balance of highly cohesive components, without

over-complicating the overhead to connect them is inherent to the nature of complex systems itself. As a result, there is little direction in literature on how exactly to evaluate components and systems under decomposition. Formalising these criteria is instrumental to automating a model decomposition process but also provides elementary insight into the quality of system decomposition. Therefore the final research question is:

RQ3: How can model decompositions be evaluated quantitatively?

1.6 Research design

Based on these research questions, a research method is developed that steers the flow and phases of this thesis. The research flow diagram in Figure 1.1 gives an overview of the main phases of this thesis and the steps taken within each phase.

Phase 1: Planning

Given the research aim, the research approach is focused on designing a new method. This starts with exploring the problem then an extensive literature review which intentionally draws on results from a wide range of academic domains. The aim is to retrieve novel and inspiring studies that have not typically been applied to model decomposition but that may prove fruitful if adapted. Equally, this phase of the research provides a solid foundation in the core theory of systems, decomposition and modularity. The research design and the selection of the use case were also conducted during this initial planning phase of the thesis.

Scope of use case

Hofmann argues that guidelines for model decomposition are needed, but that they are derived from the driving force for decomposition in every specific case [26], a sentiment that is generally accepted in literature and industry. Therefore this thesis is scoped for a specific class of models in the logistics and process modelling domain. This domain is chosen due to the rising demand for model management as Digital Twin technology leads to increasing complexity and size of models. Therefore, this class of models is an appropriate scope for researching model decomposition.

Specifically, this thesis uses the Logistics Specific and Analysis Tool (LSAT) which is described in more detail in Chapter 3. This research develops a methodology to decompose models built in the LSAT formalism. For this, two models are used; the first is a monolith model and the second is a modular model decomposed manually by an expert in the field. Therefore, the results of the automated decomposition can be compared to the manual decomposition to measure the quality of the methodology for the LSAT formalism. More generally, this acts as a proof of concept that the proposed methodology can be functional and useful for model decomposition in the class of logistics and process modelling.

Phase 2: Design

The body of the research consists of the design phase whereby the methodology is developed. Here, the three research questions are answered. Figure 1.1 shows the addition

of a fourth step in the design phase that resulted from the findings during the design of the decomposition algorithm. Each step in the design phase consists of an initial literature review followed by iterations of brainstorming, thought-experiments, exploratory implementations, trial and error, and constant reflection on the research questions.

Phase 3: Implementation
Along with the design of the decomposition methodology, intermediate design ideas are incrementally implemented in Python in order to test on source models from the use case.

Phase 4: Evaluation
It is in this phase where the method is put to the final test. By evaluating the results of the automated decomposition methodology in comparison to a manually decomposed model that was developed by an expert in the field of model based systems engineering, the proposed methodology is validated. The automated and manual decomposition are compared under a number of carefully selected criteria. It is important to note that the model that is decomposed by the expert has not been used until the proposed methodology is complete. Although it would be useful to examine the topology of the model that was manually decomposed, in order to guide the development of the automated decomposition process in Phase 2, this was intentionally refrained from until Phase 4. Therefore, the manually decomposed model is reserved for evaluation phase only so that the results can be truly valid. The second form of evaluation is a qualitative evaluation with an expert. The purpose of this face evaluation is to determine the usefulness of the proposed methodology and identify any additional insights.

1

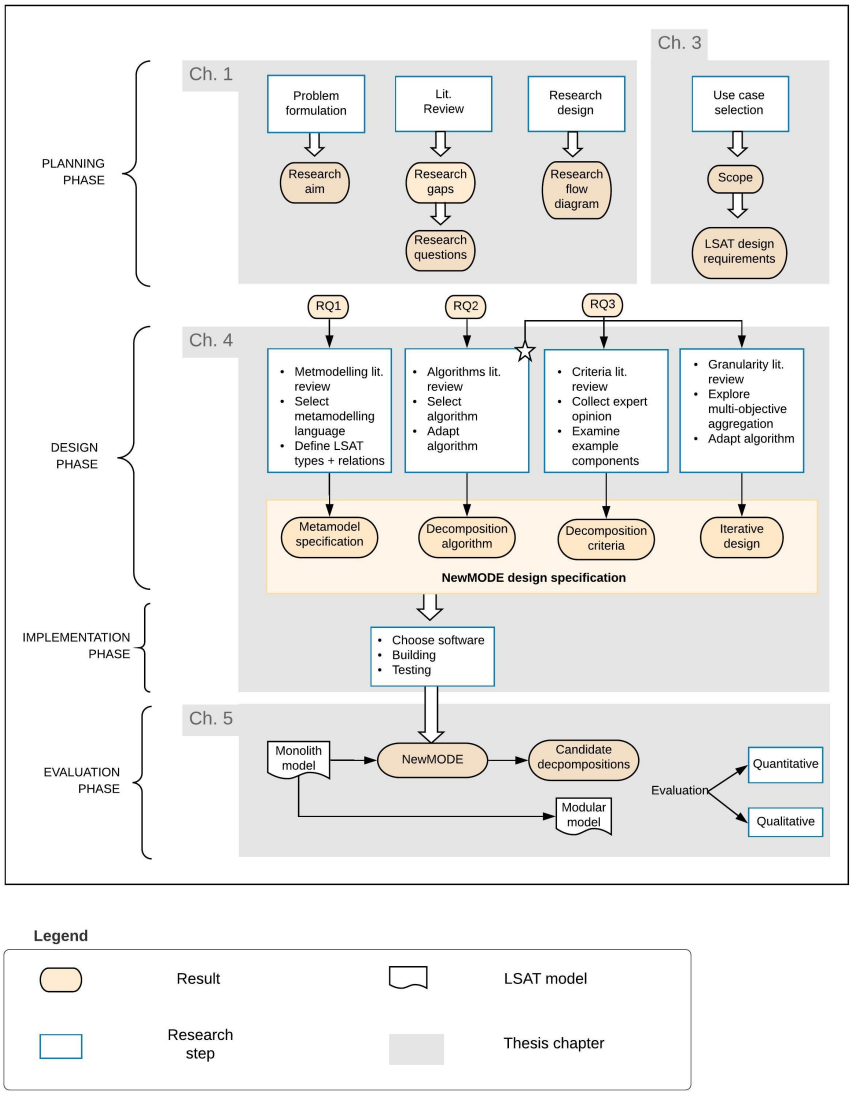


Figure 1.1: Research flow diagram.

1.7 Structure of this thesis

This thesis has the following structure:

Chapter 2 dives into the results of the literature review which is the foundation of this thesis work. Although it stands alone as a review of state of the art in topics that are related to model decomposition, its purpose is to provide reasoning to the research gap as well as the design choices made throughout this thesis. Figure 1.1 shows that the literature review was conducted during many stages of this thesis, first in the planning phase and again at the beginning of each step in the design phase.

Chapter 3 introduces LSAT, a modelling tool for which model decomposition will be implemented. A description of an example machine in a manufacturing process is used to highlight the functionality of LSAT as well as the complexity in the model.

Next, the proposed method is described in Chapter 4. This starts with an overview of the proposed methodology which is split into 4 main tasks; metamodelling, decomposition, criteria, comparison of candidates. Each of these tasks is discussed in turn, with details of the interesting challenges that arose within the design of each subprocess explained. The focus in this chapter is highlighting the reasoning and logic behind the design of the proposed methodology.

Chapter 5 corresponds to the evaluation phase in the research design. This describes the evaluation of the novel methodology by comparison to the manual decomposition completed by an expert. Important results are found in this Chapter.

In Chapter 6 there is a discussion of the limitations of this thesis as well as reflections on various aspects of the design of NewMODE. As well as this, the impact of this thesis on modellers, organisations and society is discussed.

Finally, Chapter 7 concludes this thesis with a particular focus on the answers to the research questions. The key contributions of this thesis are highlighted, followed by final suggestions for future work in this research topic.

2

Literature review

The literature review presented in this chapter serves multiple purposes. The state-of-the-art in decomposition theory provides a theoretical foundation to the topic. Novel studies from microservitisation provide real implementations of re-architecting legacy software, with inspiring and varied approaches. The literature on metamodelling and network science provides justification for scoping the methodology in that direction.

2.1 Decomposition criteria

We define decomposition criteria as the indicators that a given decomposition arrangement is successful or not. This, of course, is a logical place to start when attempting to design a novel method for model decomposition. This refers to the (un)desirable qualities of both the decomposed system and the individual components and how those qualities might be specified. In studies from both M&S and SWEng this includes 'high cohesion and loose coupling' [27], [4], [28]. SySci literature is more specific when it comes to defining these terms, considering decomposition is at the core of SySci. Although many definitions of loose coupling exist, they all find their origin in Simon's paper on "The Art of Complexity" from 1962. His advice for defining subsystem boundaries is yet to see considerable improvement:

> "In general, the critical consideration is the extent to which interaction between two (or a few) subsystems excludes interaction of these subsystems with the others." [14]

Just a few years after, we find a clear description of high cohesion in Gauthier & Stephen's book:

> "Each task forms a separate, distinct program module. At implementation time each module and its inputs and outputs are well-defined, there is no confusion in the intended interface with other system modules." [29]

Today, the most often quoted guideline for system decomposition in SWEng is the Single Responsibility Principle (SRP), named by Robert Martin who states that classes should be built such that each "has only one reason to change" [30]. Although these

concepts remain unchanged for decades, literature that specifies their implementation are hard to find and most studies do not evaluate the degree of cohesion or coupling for the model or individual components exactly. The literature presented here is the state of the art in studies that have made contributions to such.

A categorical evaluation.
Stevens, Meyers and Constantine introduce the concept of architectural mismatch which

> "stems from mismatched assumptions a reusable part makes about the structure of the system it is to be part of. These assumptions often conflict with the assumptions of other parts and are almost always implicit, making them extremely difficult to analyse before building the system." [31]

Of the three theoretical foundations aforementioned this is most closely related to the SRP. They use this as motivation to strive for high cohesion and loose coupling of components. The authors specified an ordinal scale for evaluating components in terms of coupling and cohesion with a clear focus on the individual components rather than system criteria. Although they provide a useful scale for evaluating coupling as the 'degree to which two modules communicate' and cohesion as 'the degree to which a module provides only one functionality', the link to their original motivation for reducing architectural mismatch is unclear. Hofmann discusses the decomposition of a system by grouping components according to the modelling assumptions on which they rely [26], although the criteria for this are not explicit.

A metric-based evaluation.
The service cutter framework is a user-friendly tool with 16 criteria and a handful of algorithms for decomposition that can be selected from by the user decoupling criteria [32]. The monolith must be manually pre-configured to fit the unconventional input syntax required. Extending from Gysel's 16 coupling criteria , a metric-based evaluation framework for microservice decomposition is developed by Taibi [7], summarised in Figure 2.1. This framework specifies formulae for the decomposition metrics defined on an architectural level, with maintainability as the driving force of decomposition. The limitation to their implementation is the assumption that a detailed log file of the monolith is available with complete execution path for all methods and classes. This provides specification for evaluating decomposition criteria and covers a variety of system qualities that capture the trade-off between performance and maintainability. The main limitation of this approach is its static nature. The frequency and temporal distribution of inter-component communication is not evaluated, and this is an important aspect of simulation performance that separates it from web-based software applications.

Granularity exploration
As mentioned in Chapter 1, a known challenge of decomposition is the tradeoff between size and number of components [33]. The decomposition granularity refers to the number of components in the decomposed architecture and is known as one of the key design choices that influences the quality of services in software [21]. Hassan, Ali and Bahsoon proposed a novel approach to modelling granularity of software applications using ambient

Criteria	Definition	Level	Calculation
Coupling	Coupling between microservices (CBM)	Component	(Number of external links) / (Number of classes in the microservice)
Granularity	Classes per microservices (CLA)	Component	Number of classes
Duplication	Number of duplicated classes (DUP)	System	Number of classes that exist more than once in the system
Overhead	External calls (FEC)	Component	(Number of weighted call instances) / (Number of classes in the MS)

Figure 2.1: A Metric-based evaluation framework for microservices decomposition, based on [7].

algebra. They focus on performance of the resulting application and the organisational aspects of software development tools to investigate patterns for granularity. Otherwise, studies that do not expect granularity as a user defined input are sparse.

Overall, the concept of model decomposition is long standing and that it is generally accepted that a good decomposition strives for high cohesion and loose coupling. When it comes to translating this into meaningful criteria that can be used to quantitatively evaluate or compare different decompositions, the literature is sparse. Equally, the question of granularity in decomposition is a known challenge, with very few studies considering it within the scope of analysis.

2.2 Understanding the monolith

In order to decompose a model in a reproducible manner, a formal representation of it is needed. Gaining understanding of a model is typically done by manually exploring the code, its input-output behaviour, reading design artefacts or asking the developer. Considering time constraints and reproducibility there is a need to consider techniques that generate a model specification from the legacy code itself. This relates directly to RQ1 in this thesis and the literature in this subsection aim to provide the foundation for answering the question of how to represent the monolith model. First, we review two academic domains that focus on representations of models; Model Driven Reverse Engineering (MDRE) and Model Driven Development (MDD). Figure 2.2 illustrates the place of MDRE and MDD in the fields of SWEng and MS, where the arrow follows the direction of automated generation, and the metamodel refers to a higher-level specification of the legacy code.

Within SWEng, the need to understand legacy applications is addressed by MDRE, a concept defined by Rugaber and Stirewalt as:

> "the process of understanding software and creating a model, suitable for documentation, maintenance or re-engineering." [34]

This is represented by the arrow on the left-hand side of the illustration. On this theoretical foundation there are several studies with the aim of understanding complex software applications. Schneider & Koziolek proposed an automated code-to-metamodel method

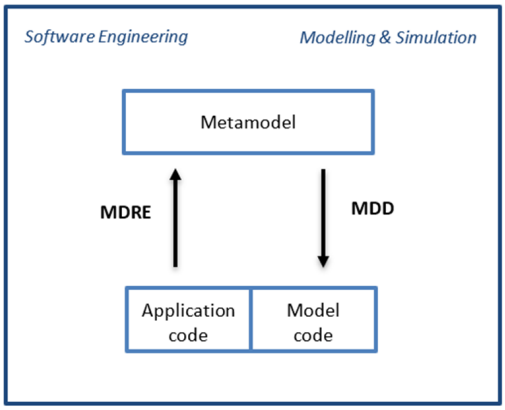


Figure 2.2: A comparison of Model Driven Reverse Engineering and Model Driven Development.

that matches chunks of code to components from domain specific libraries [35]. Although this does not consider simulation models specifically the approach might be applicable for simulation languages with available model libraries. The open source MoDisco is proposed by Bruneliere to improve the ease of implementing MDRE for large, legacy systems [25]. Escobar et al. proposed a variant of UML for specifying high-level architecture of java code, using MoDisco [36]. Overall, these MDRE techniques are well-automated and reproducible. However, they are developed for web-based applications and simulation models are out of their scope. Therefore, they would need modifications to be used in this research to distinguish functional versus execution logic of the simulation.

In the field of M&S, metamodelling has been used as a way to describe models during model driven development (MDD) [5]. In this application, a metamodel is defined as a model of a model whereby it "describes the possible structures which can be expressed in the language" [20]. As this technique is typically used during the development of models from scratch it has a top-down approach, as illustrated in figure 2.2. This requires designing a metamodel upon which transformations can be applied, the last of which results in source code for execution [20]. Although there are implementations for metamodel-to-model, model-to-model (M2M) and model-to-code (M2C) transformations, code-to-model generation is not considered explicitly. Cetinkaya & Verbraeck define criteria for model transformations, one of which is 'reversibility'. Therefore, it might be possible to reverse M2C transformations to automatically generate metamodels from code even though this was not the original intention. This would imply reversing the direction of the arrow on the righthand side of the illustration.

The use of metamodelling in both MDD and MDRE is equivalent in their intuition. It allows a user to arrange the structure of model/application code by applying transformations on the metamodel that result in a new metamodel with different topology. The user can experiment with different topologies or transformations before changing the code. The first type of transformation is classified as a horizontal transformation as the resulting metamodel is in the same level of abstraction as the original metamodel [5]. This is distinct from vertical transformations which, when applied to a model, result in a model of lower

or higher abstraction. Specifically, this means converting from model to metamodel, or from a metamodel to a model, and is equivalent to the arrows shown in Figure 2.2.

In summary, the theory of MDRE and MDD supports the choice to reason about the architecture of model code, on a higher level of abstraction; specifically with a metamodel. Although the literature supports this choice, there is still a clear gap when it comes to the purpose of this thesis. In Figure 2.2, the transformation from Model Code to metamodel is absent and although it is theoretically possible to reverse the metamodel-to-code arrow, there is no known implementation of this vertical transformation for the class of models we consider.

2.3 Migration from mononolith to microservices

Due to the lack of studies that have implemented a method for migrating a monolith simulation model to a modular design, other literature from SWEng is examined. As mentioned in Chapter 1, the Microservices architecture has recently emerged as an architectural standard for managing complexity of software systems by structuring software into small, loosely coupled components. Due to the high priority of migrating legacy systems into microservices, there is a wealth of literature on the decomposition of software systems for microservices. Although many of the systems differ from the class of model under consideration, the fundamental objective is aligned and the intuition of many studies is enlightening.

Microservices patterns

Richardson's book on microservices patterns provides a detailed set of patterns which are generally described as "reusable solutions to problem that occurs in a particular context" [3]. Four strategies are provided for decomposing legacy applications into microservices:

- Decomposition by business domain
- Decomposition by subdomain
- Self-contained service
- Service per team

Many of these strategies rely on access to process models of the business. There is a recent trend in storing patterns in open source repositories to share decomposition strategies amongst developers. Djogic et al. have provided a step-by-step framework that employs existing patterns to migrate an event-driven service oriented platform to a microservices architecture. Wrappers are applied to each microservice and the primary objective is to achieve logically independent components [37]. In general, the challenge remains of defining patterns that are finely grained enough to create customized decomposition plans for specific use-cases but that remain generalizable from the use-cases they were identified. A Variability-based, Pattern-driven Architecture Migration (V-PAM) method is proposed by Jamshidi et al. to facilitate organizations in selecting appropriate migration patterns (from existing libraries), incrementally build patterns to form a migration plan and extend this based on specific patterns identified from the process model of the monolith system [38].

This framework adds flexibility to the pattern based approach to decomposition. The interesting contribution of microservices patterns is the formal specification of decomposition solutions.

There are also many microservices patterns based on interface analysis techniques. The idea here is to map functions that contain frequent references of parameter names together, and then optimize the granularity of the configuration [39]. Relying on semantic similarity described through OpenAPI specifications, promising results have been shown for domain agnostic techniques. The fundamental assumption is that components with similar vocabulary contain similar functionality. In general, this group of techniques depend on well-defined interfaces, that have been written according to (sometimes strict) coding conventions.

Cross-tier dependency graphs.
Levcovitz et al utilise 'dependency graphs' to propose candidate solutions with loose coupling and prioritise the clustering of data tables with their dependent functionality [40]. The main limitation is the requirement that monolithic applications have a structure of smaller systems where each subsystem has a well-defined set of logical functionalities and its own data store already. Thereafter, graph theory calculations can be applied to optimize the decomposition.

Data flow driven.
One of the most novel techniques in recent years makes use of Data Flow Diagrams (DFDs) of an existing monolith to decompose the model according to how much communication flows between each component. Clustering algorithms are then applied to these data-flows to find the optimal configuration [28]. Here the decomposition criteria are to minimize data flow between components only. This relies on the existence (or manual creation) of a reliable DFD as well manual intervention by the developer.

Unsupervised learning decomposition.
Using the application access logs and an unsupervised machine-learning method to automatically decompose the application into microservices, Abdullah et al. achieved reasonable performance in resultant microservices systems [17]. The information from the access logs allow partitioning of the model into microservices that have similar performance and resource requirements. Automation is high by only optimizing on performance of candidate configurations, and integrating the technique with existing automatic deployment and scaling tools. The fundamental limitation of this tool is general of all unsupervised learning applications in that the decomposed components lack interpretability. Not only is this less reliable in complex systems which are often irregular or unpredictable, but this negates the motivations for decomposition in the first place; maintainability and reusability.

Metamodelling for microservices identification
Escobar et al., developed a process to help software development teams in understanding monolith applications in order to facilitate transformation to microservices [36]. They provide an approach that visualises the architecture of legacy software applications as a set of diagrams, generated from the application itself. The purpose of these diagrams is to aid designers in understanding a monolith software application that has no design

documentation or has gradually evolved without any restructuring of the code. The second contribution of these diagrams is that they can be used to propose partitions of the application that might be suitable for microservices migration. The intention is to provide suggested partitions as a starting point to system decomposition for software developers. Although the intuition of this study is related to this thesis, it relies on two design choices that are not applicable to the scope of this thesis. Firstly, the visualisations are implemented for Java Enterprise Edition (JEE) platforms and rely fundamentally on the idea of separating Enterprise Java Beans (EJBs). Secondly, the main challenge for decomposition of these kind of software applications is the partition of the database and the partitions are based on a *separation by data* approach. The scope of the application does not consider challenges in simulation models such as co-simulation and inter-component synchronisation during execution.

Overall, the SWEng offers many interesting developments in migration from monolith to microservices. Approaches are varied and successful. However, the focus is specific to enterprise software applications, typically web-based and service-oriented in their 'monolith' version. Although these approaches are inspiring, it is unlikely that they can be applied to simulation models directly.

2.4 Metamodelling formalisms

As discussed in Section 2.2, a metamodelling approach allows reasoning on the architectural level. Considering that metamodels are models too [20], a formalism for metamodelling must be chosen, if this is to answer RQ1. Different types of metamodelling approaches were investigated as summarised in Table 2.1 which states the options for describing dynamic behaviour within a model and possible tools for each formalism. The chosen techniques were identified from the examples of microservices migration as discussed in Section 2.3.

Table 2.1: An overveiw of metamodelling formalisms explored.

Metamodel formalism	Behaviour modelling	Tools
UML	Activity diagrams	fREX, pyEcore, Src2Mof
	Sequence diagrams	
State-machines	Executable state machine	sparx systems
Graph theory	Dynamic graphs	networkx, neo4Java
	Versioning history	
	Directed Acyclic Graph	

UML is an established modelling language and it would certainly be convenient for representing the static dependencies of a model. However, in order to account for dynamic behaviour of the model, activity diagrams would be needed which are unscalable, considering they are not designed for runtime analysis but for physical activities [41]. The dynamic behaviour of the model is a key factor for its high complexity and so it is essential that the metamodel can account for more than just the static architecture. Specifically, the metamodel needs to distinguish between concurrent and sequential execution of functions and this is not native to UML.

Taking a different perspective, the possibility of representing the behavior as a state machine holds potential [42]. This type of metamodel would make representing the variance in execution pathways convenient. For example, one of the challenges in simulation model decomposition is that the stochasicity can lead to an infinite number of execution paths. Therefore, it is challenging to propose a decomposition architecture that performs well for different execution paths. A state machine can represent static architecture and the dynamic behaviour by describing the transitions between different assigned states. However, this would result in a large number of states that is likely to be computationally expensive, if a holistic profile of the execution paths is to be captured [42]. After discussion with experts, it was determined that this variation in execution paths is potentially relevant to exploratory modelling during the design phase, but it is unlikely for an operational model in the manufacturing industry, as is the case for Digital Twins. Therefore, the benefits offered by state machines are not relevant for the use case. Furthermore, this form of metamodelling imposes restrictions on modelling the static dependencies which is a key requirement also.

Finally, it is determined that a graph based approach offers the most flexibility, scalability and emphasizes the interactions between components above all. Its main advantage is that **interactions are considered as first class entities** in this formalism, as edges between nodes. Edges are as important as nodes, and can be accessed by themselves directly, without first accessing the nodes they are connected to. This has proven to be powerful in many applications. In 1736, Euler proposed a graph representation of the city of Konigsberg which made previous route calculations almost trivial under the graph-based model [43]. In addition to the suitability of the formalism, the extensive tools and theoretical developments to graph theory offer flexibility and scalability. One important sub-part of graph theory is network theory, which can be seen as the study of graphs in the real world whereby nodes and edges possess attributes [44]. These attributes allow a metamodel to have many types of relationships, some of which can be borrowed from UML for example.

2.5 Graph and network theory

Section 2.4 highlights the reasoning for preferring a graph-based approach for metamodelling. This is supported by the studies in Section 2.3 that incorporated a graph-based approach, albeit in a different application. If this is to be an appropriate start for metamodelling, it must also support the answering of RQ2. Therefore, below is a discussion of potential decomposition algorithms that exist in the domain of network science. Different types of decomposition algorithms were considered and are discussed below. The chosen techniques were identified from a starting point of the decomposition approaches found in literature that was mentioned in Section 2.3. In network science, a community can be defined as a subset of nodes within a graph such that connections between the nodes of a community are denser than external connections to nodes in the rest of the network.

Agglomerative

One class of the so-called hierarchical clustering algorithms is the agglomerative approach [45]. For every pair of nodes u,v in the network, an edge (u,v) is created with a weight measuring how closely connected u and v are. Starting with a network of all the nodes and no edges, links are iteratively added between pairs of nodes in order of decreasing

weight. In this way nodes are grouped into larger and larger communities. Algorithms of this kind are called agglomerative. A popular agglomerative approach is referred to as 'modularity maximization' techniques. The number of edges inside a component are compared with the expected number of edges that would be found in that cluster, if the network were a random network (null model) with the same number of nodes, where each node keeps its degree of edges, but edges are otherwise randomly attached. In order to apply this for a directed, acyclic graph as is our case, their needs to be a meaningful null model for DAGs to use as the comparison. In [46] they test different null models to use a 'modularity maximization' approach for a DAG network, but return no conclusive results. However, this approach suffers from the resolution limit in that it does not find small communities. Therefore, the modularity maximization approach is not well established for directed graphs and the inability to find small communities is undesirable for this thesis.

Minimum Spanning Trees

Another branch of techniques make use of the Minimum Spanning Tree (MST) to implement decomposition. For example, focuses on algorithmic extraction of microservices from a monolith java, web-based application [33]. The approach is to create a weighted graph that represents the versioning history, from Git, of this web-based application. Although this has shown excellent results for their study, a DAG cannot be fully represented by an MST as this would essentially remove any concurrency for which we made the DAG in the first place. Therefore the MST-based approaches are not meaningful for this thesis.

Divisive algorithms

For the other class of algorithms, called divisive, the order of construction of the decomposition is reversed: starting with the whole graph and iteratively removing the edges, progressively divides the network into smaller disconnected components, which are identified as communities. The crucial design choice in a divisive algorithm is the function that selects the next edge to be removed, which aims to find edges connecting communities and not those within them. In 2002, Girvan and Newman (GN) introduced a divisive algorithm where the selection of the edges to be cut is based on the value of their 'edge betweenness' [47], a generalization of the well established measure of centrality betweenness defined for nodes in a network. Each edge in the network can be associated with an edge betweenness centrality value. In each iteration of the GN algorithm this function is recalculated for all edges, and the edge with the highest betweeness centrality is removed. It is important that the next edge for deletion is recomputed in each iteration. Although it might be tempting to rank the edges and store this for the entire run, this strategy can fail. If two communities are connected by more than one edge, then there is no guarantee that all of those edges will have high betweenness; it is only certain that at least one of those edges will have high betweeness. By recalculating betweennesses after the removal of each edge it is ensured that at least one of the remaining edges between two communities always has a high value.

The GN algorithm represents a major step forward for the detection of communities in networks, since it avoids many of the shortcomings of traditional methods [45]. In addition, it is possible to adapt the function that selects the next edge for removal such that it is meaningful for directed networks which makes it an obvious choice for this thesis work.

2.6 Summary

Overall, there is a convergence in literature from different academic fields that a good decomposition possesses high cohesion and loose coupling. Contrastingly, there is very sparse literature in determining how to implement or interpret these criteria quantitatively. Similarly, there is very little attention in literature to determining a stopping condition for decomposition, with very few studies considering it within the scope of analysis but rather requesting the number of components as a user defined input. The theory of MDRE and MDD supports the choice to reason about the architecture of model code, on a higher level of abstraction and contextualises the use of metamodelling in this domain. There is a gap in transformations from model code to metamodel, although the equivalent exists for many software applications. Overall, the SWEng offers many interesting developments in migration from monolith to microservices where the original monolith is already service-oriented and a web-based application. Many studies consider the partition of data as a main challenge in system decomposition. Although these approaches cannot be applied directly to simulation models, they inspire the use of graph theory and metamodelling for representing the legacy system. One of the frequently occuring approaches in microservices migration is the use of a graph-based metamodel. This is supported by its comparison to UML Activity Diagrams and State Machines. It offers flexibility, powerful analysis and fundamentally models interactions which are at the forefront in the question of decomposition. If this graph-based approach is to be an appropriate formalism for metamodelling, it must also support the answering of RQ2. Fortunately, the field of network science has existing techniques in community detection in social networks. The intuition is very similar to model decomposition and there is evidence that the Girvan Newman algorithm is suitable for a directed graph.

3

Logistics and process modelling

Throughout this research, we use a case study from the design of manufacturing systems domain. As high-tech manufacturing systems become more advanced and Digital Twin technology emerges, there is an increasing use of logistics and process models for design of complex manufacturing systems. Specifically, we consider a formal modelling approach which is supported by the Logistics Specification and Analysis Tool (LSAT). LSAT has been developed by ASML, TNO's Embedded Systems Innovation (ESI) group, and Eindhoven University of Technology [48].

3.1 An example machine

As an illustrative example, let us consider a bitesize section of a complex manufacturing process. This serves as a running example throughout this thesis to demonstrate the concepts and methods applied. This machine illustrated in Figure 3.1 uses 3 robotic arms to perform an operation on a very small product. This machine fist into a large production line and the product gets taken from the previous stage, then operated on before it is passed onto the next machine in the production system. As an example, we will consider this machine is built as a monolith model to analyse its behaviour. Robot A puts the product onto the workholder, then the workholder rotates 90 degrees anti-clockwise. Next, Robot B performs an operation on the product, which is now mounted on the workholder. Again, the workholder rotates 90 degrees anti-clockwise so that the camera can inspect the quality of the work done by Robot B. Finally, another 90 degree rotation happens and then Robot C, removes the completed product and transfers it to the next stage of production.

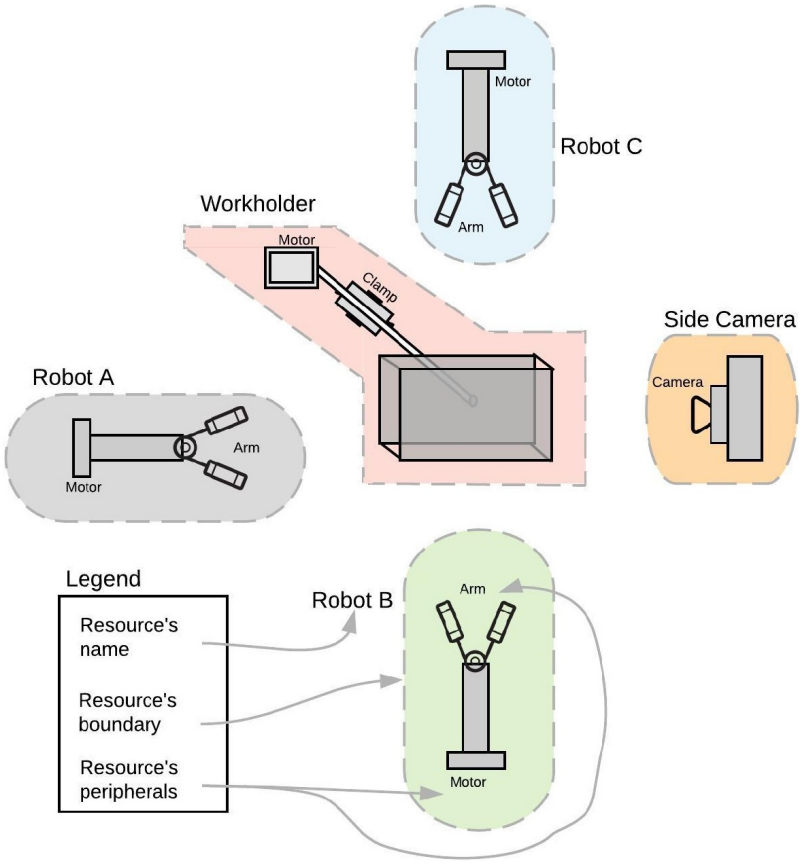


Figure 3.1: Diagram of machine with 3 robots, a camera and a workholder.

Even without considering what happens before or after this part of production, it is a complex model due to the inter-dependencies between resources. For maximum throughput, some of these actions are happening concurrently, so that 4 different products are being processed by a single workholder at any one time. The dependency of each action on other actions is fully illustrated for the entire activity in the Appendix. In Figure 3.2, a high level illustration of this activity is shown. Each swimlane represents one of the resources and their actions, and black bars represent synchronisation points between actions of different resources. For example, while the workholder is rotating 90 degrees, the Robot A can already be collecting the next product. When the workholder has stopped rotating, and the clamp has fixed it in place, Robot A is ready to place that product onto the workholder without delay. The camera's inspection action takes longer than the actions of the Robotic arms. Although the camera must wait for the workholder to finish rotating, it does not need to wait for the workholder to be clamped or unclamped as it does not apply any force to the workholder. Therefore, these actions can overlap if needed. The subset of this diagram that is outlined by the dashed line will be used as an example throughout Chapter 4 to demonstrate the proposed methodology.

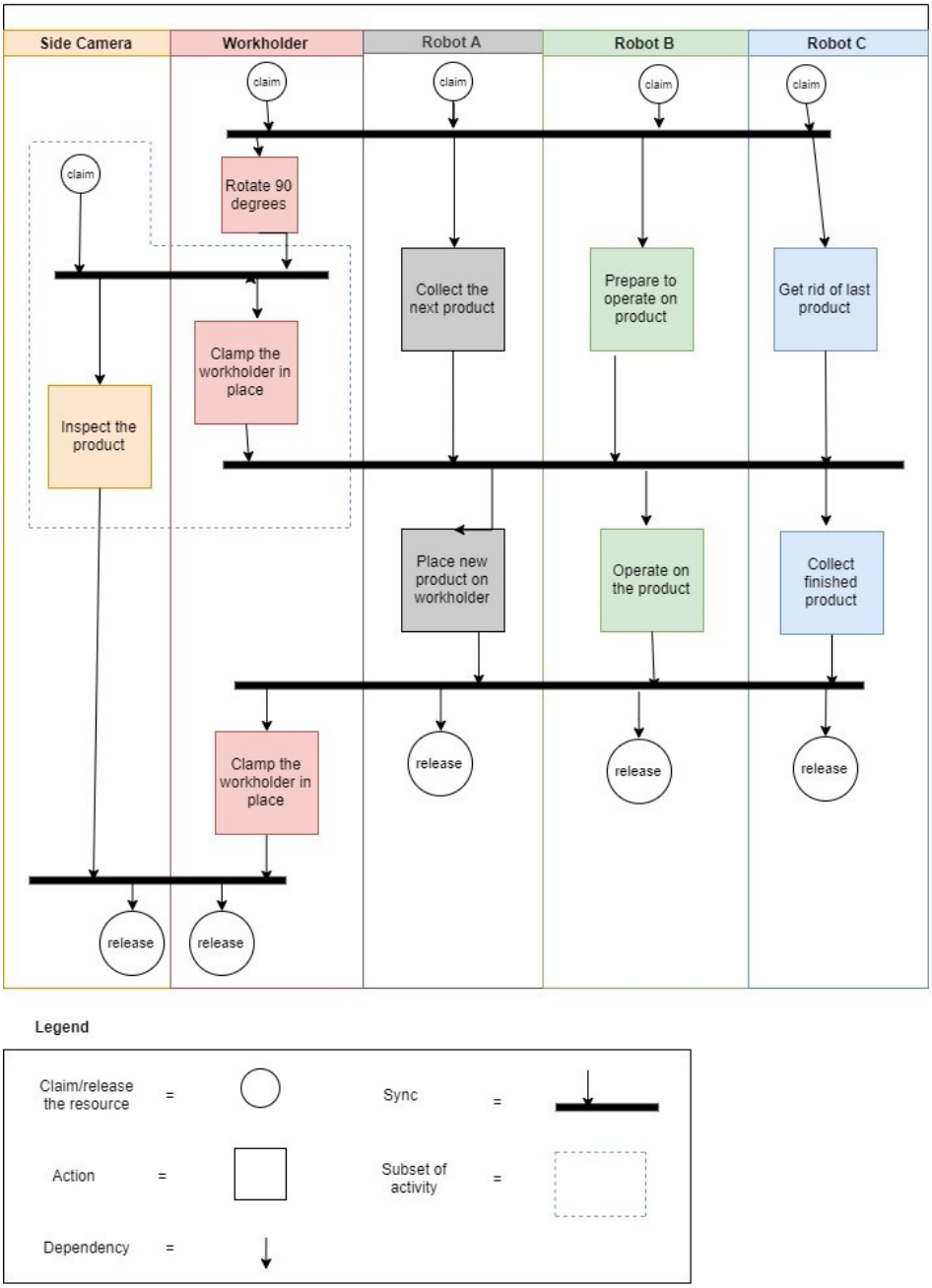


Figure 3.2: The activity flow diagram.

3.2 LSAT primer

In the section below, there is a brief explanation of some of the key concepts used in LSAT; machines, activities, dispatches.

3.2.1 Machines

A machine is specified in LSAT using resources, peripherals and actions. Resources are physical parts of the machine, like the robots or workholder illustrated above. Each of these resources has a number of subparts, called peripherals, and actions are specified for each peripheral. For example, a robot is a resource that has two parts: a robotic arm (that can grab and release products) and a motor (that moves the robot). These actions take time that can be specified as stochastic delays in the LSAT model.

3.2.2 Activities

Actions are the base unit in the model and can represent microtasks such as gripping an product. Therefore, a manufacturing process is made of a large number of LSAT actions[1]. Certain actions must precede other actions sequentially whereas other actions can be performed in parallel. The dependency between actions is specified as an Activity in LSAT. The diagram shown in Figure 3.2 is an example of a single activity. Considering the size and complexity of a realistic manufacturing process, it is not very user-friendly to input an entire manufacturing process as a single activity. To ease the burden of inputting these tasks, standard practise is to divide the manufacturing process into a number of managable subparts, and then create an activity for each of these in LSAT. The model we consider has been specified as 5 activities. Currently, the choice of how to aggregate actions into activities is a design choice [49]. **In LSAT, decomposition refers to dividing a model into a number of activities** which consist of actions and their dependencies.

3.2.3 Dispatches

To analyse the system, a dispatch file needs to be created. A dispatch defines a sequence of activities that should be performed in order. A single dispatch file might have dozens of activities, which make up the manufacturing process. In order to experiment with the manufacturing process design, users can specify different dispatches and compare the throughput.

3.3 Driving forces for decomposition

The aim of an LSAT model is to analyse the physical system using the specified model, in order to optimize the design of the manufacturing system. The model may also be used to detect anomalies in the real system i.e. when the throughput of the real system deviates from that calculated by the model. Based on the purpose of this model, its decomposition can be considered according to Hofmann's driving forces [26] as:

- increase the range of manageable complexity in system design,
- increase the range of manageable complexity in system analysis,

[1] To be precise, this should be named as action **instances**, a term which will be introduced in Chapter 4. Until then, the terms action and action instance will be used interchangeably.

- improve maintainability and consequently the tolerance versus uncertainty,
- decrease redundancy within a design/implementation,
- increase reuse between different designs/implementations.

3.4 Summary

This thesis is scoped for a specific class of models in the logistics and process modelling domain. This domain is chosen due to the rising demand for model management as Digital Twin technology leads to increasing complexity and size of models. Therefore, this class of models is an appropriate scope for researching model decomposition, especially based on the identified driving forces for model decomposition. This thesis specifically addresses decomposition of models built in the LSAT formalism. Here, a model is comprised of machine, activities and dispatches in LSAT and decomposition refers to grouping action instances into activities.

4

The design of NewMODE

The main contribution of this thesis is a novel approach to model decomposition that is analytical, semi-automated and reproducible. In order to achieve this, the following steps are performed, as illustrated in 4.1:

1. Define the metamodel.
2. Design the decomposition process.
3. Specify the criteria for evaluation.
4. Compare the candidate decompositions.

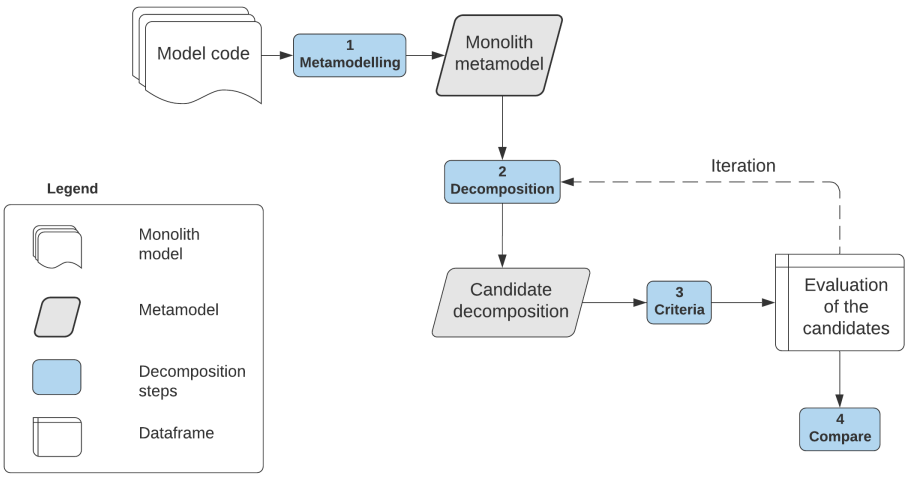


Figure 4.1: Overview of NewMODE steps.

The task of metamodelling, decomposition and criteria each address one of the research questions directly. During the design of these steps it became apparent that a fourth comparison step was needed in order to address the granularity trade-off in model decomposition.

First, a formal representation of the model architecture is sought. For a complex system it is a laborious and non-trivial task to generate a metamodel without any automation. By formalising this representation, it is possible to automate the transformation from model to metamodel for specific source formalisms. Using this it is possible to decompose the metamodel in an automated fashion and present candidate architectures. Although it is true that automating this process increase the usability of this approach, it is equally motivated by the desire to more extensively explore the possibility space. For the models being considered in this thesis, it is unrealistic to perform multiple decompositions without automation. The next task is to evaluate these candidate architectures in a way that facilitates comparison for modellers. As shown in figure 4.1, decomposition and criteria are applied iteratively to explore the possible decomposition architectures. The methodology for each of these tasks is novel and many design choices need to be made. The rest of the chapter describes the design of these steps followed by a brief description of the implementation of NewMODE.

4.1 Metamodelling: representing the model architecture

As discussed in the problem statement, a model-driven approach allows reasoning on the architectural level which leads to the first research question:

RQ1: How can the architecture of a complex model be represented?

Considering the advantages and disadvantages of the various metamodelling formalisms reviewed in 2.4, I choose a network based approach to metamodelling of the model architecture. By using the networkx package in Python it is possible to specify a directed, multi-edge, weighted graph. This network has three layers to represent the different aspects of the model. Layer 1 describes the possible actions of the machine as static dependencies. Layer 2 describes the dynamic execution that occurs sequentially or concurrently between different model actions. These layers are connected using relationships defined by UML. The dynamic behavior is modelled as an execution DAG with forks and joins representing concurrent execution. These DAGs can be coded from the model code directly and dose not require runtime analysis. A third layer of the network specifies the decomposition architecture. This layer describes the contents of each component. The architectural metamodel is specified as a graph G that has vertices with labels, and arcs connect pairs of vertices. Arcs have a direction and a type so there can be multiple edges between any pair of vertices to represent different types of connections. Labels are used to define different types of vertices and different types of arcs. Details of the metamodel specification are given in the next three subsections, layer by layer.

4.1.1 Layer 1: static dependencies

The information in this layer of the model is described in the machine file in LSAT where the physical resources along with their peripherals and possible actions are specified.

In descriptions for manufacturing process (including LSAT), a machine is described as a *resource* that has multiple parts which are referred to as *peripherals* which are subparts of the resource that can execute an *action.* As described in Section 3.1, the example manufacturing process has three robots which are each a single resource. The robot consists of two peripherals: the motor and the robotic arm. Actions are defined for each peripheral, and the complete set of actions describes all the behaviour that the robot can execute. For example, the robot in the running example can be described as follows:

- **Resource:** this is the physical robot.
- **Peripheral:** the robot has two peripherals: its motor and its robotic arm.
- **Actions:** are described for each peripheral. The motor can move the robot to different positions, it can extend slightly or retract away from the workholder for precision movements. The robotic arm can collect or deliver objects for the robot.

These things are all specified as nodes in the metamodel, and designated a label with their node type as 'resource', 'peripheral', or 'action'. Each resource is then connected to its peripherals with an edge, and each peripheral is connected to its actions with an edge. Specifically, these edges represent an **aggregation** relationship, as is defined in UML. This means that a resource owns its peripherals and each peripheral owns its corresponding actions. Following UML standard, the actions are described as being aggregated into their peripheral. Therefore, the edge connecting these nodes is directed from actions to peripherals and is identified with the type aggregation.

The appendix B.1 has a full specification of the metamodel for this machine. Figure 4.2 is a high level illustration of how the actions and resources of the example machine are described in the metamodel's first layer. There is one resource node for each of the resources, coloured with the same code as in Chapter 3. Each resource is connected with its possible actions. The numbers close to the actions signify the peripheral that performs each actions. Notice the similarity in the definition of the three robots; they all use the same actions and peripherals (motor and arm). Of course, this is not a surprise considering the description of their behaviour in Chapter 3. To avoid repetition of code, LSAT actually defines another object called Peripheral Types. For example, the Arm is defined as a peripheral Type with actions clamp and release. Then each of Robot A, Robot B and Robot C have Peripherals which are instances of these Peripheral Types. This is convenient for re-use of code for similar resources, which is common in these types of machinery.

4.1.2 Layer 2: execution behavior

The information in this layer of the model is described in an activity file in LSAT. In manufacturing systems, actions are executed in a specific order. In general, an activity is a set of actions and the dependencies between those actions. Firstly, *resources* are physical machines that must be claimed before any of its actions can be executed. Identifying the claim and release of resources is used to specify the control between machine parts for different activities. If an action C is dependent on other actions A and B it means actions A

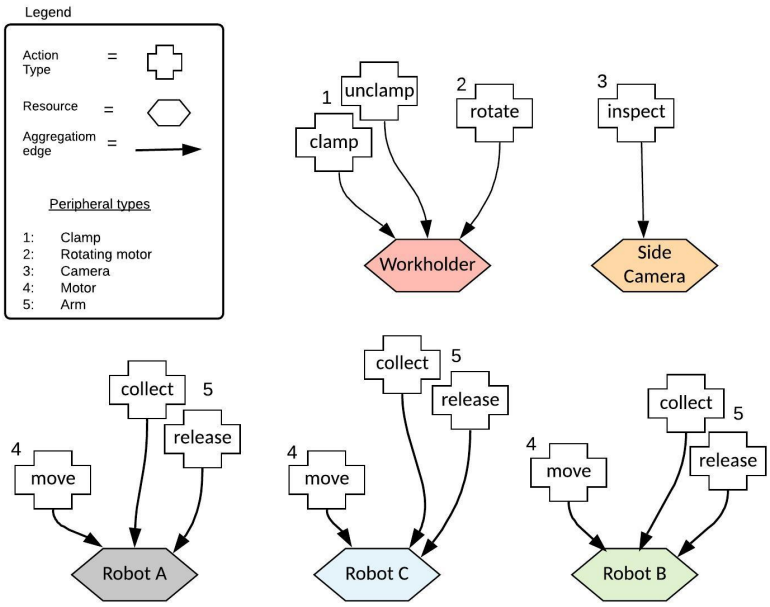


Figure 4.2: Example of layer 1 in the architectural metamodel

and B must be complete before C can occur. The dependency between action instances is identified by an edge with type *sequence*. The dependency of actions across different resources is represented by synchronisation bars. In Figure 4.3, we can see the metamodel of a subset of the actions from the example activity diagram in Figure 3.2. The sync node shows the dependency between the actions of the Side Camera (yellow) resource and the Workholder resource (pink): both the motor rotation and the claim of the Side Camera must be completed before the latter actions can be performed. Technically, this type of a graph is directed acyclic graph (DAG) because the edges have a direction and there can be no cycles in the sequence.

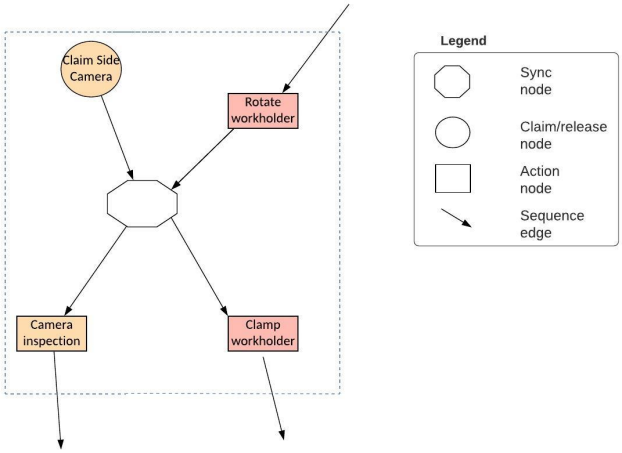


Figure 4.3: An example of layer 2 in the metamodel.

In Layer 1, an action node defined the possible actions that can be executed. In Layer 2, we create nodes that are *instances of these actions*. In addition, we define a type of node that represents the claiming of a resource and the release of a resource within an activity. Therefore the node types in this layer of the architectural metamodel are identified by the following labels:

- **Resource**
- **Action instance**
- **Claim**
- **Release**

The example activity given for this machine in Appendix A.1 represents a single activity in LSAT. There are actually multiple different activities that can be executed by the machines and coded in the LSAT model. This is used to specify different ordering of actions and to test different arrangements of the process. The question of decomposition is to identify the boundaries between activities, so it is important to specify what an activity really is by defining its properties:

- This DAG can be considered as a subgraph of the entire metamodel and represents one activity only.
- It contains only the vertices of type *action instance*, *claim*, *release*.
- For each resource in an activity DAG, a single claim vertex must precede the action instances which must be succeeded by a single release vertex.

In order to connect this layer to the vertices we specified in layer1 we define the arc labelled *instance-of*. This describes the relationship between *Action* vertex (introduced in Layer 1) and the *Action instance* vertex, as defined in UML. This is directed from the instance to the action as shows in Figure 4.4.

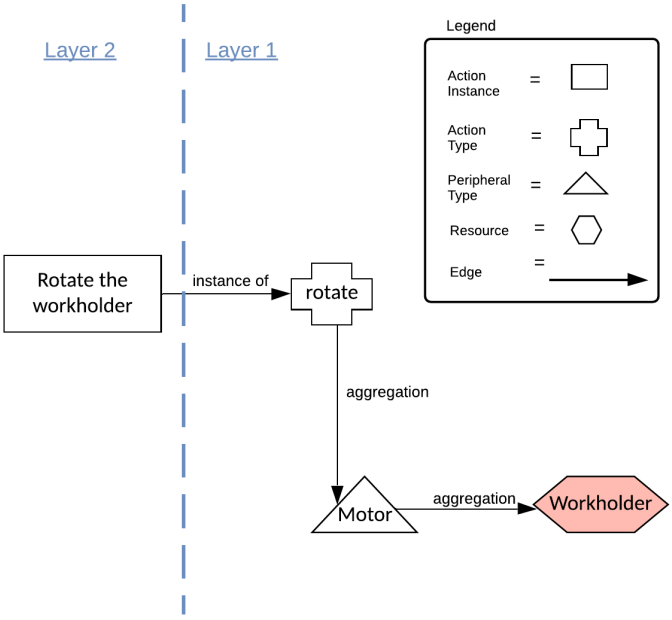


Figure 4.4: Illustrating the connection between layer 1 and layer 2 for a single action instance; 'rotate the workholder'.

4.1.3 Layer 3: decomposition architecture

A dispatch file in LSAT specifies activities to be performed as well as the order of those activities. Therefore, we represent the dispatch file as a set of vertices with type *Activity instance*. These vertices are connected by arcs of type *Dispatch Order*. In general, there can be multiple instances of any activity in a dispatch file and the order is always sequential. To connect layer 3 with layer 2, an arc of type *instance-of* is connected from the *activity instance* node to the corresponding *activity* node defined in Layer 2. Using this, the activities instances can be replaced by the directed acyclic graph that represents that activity. In this sense, it is possible *to concatenate the activity DAGs* in the order that they are specified in the dispatch file. For example, consider the two separate activities displayed in Figure 4.5 (a). Activity 1 is a version of the activity described for the example machine in Figure A.1. Activity 2 is a small activity executed by the same machine to readjust the rotation of the workholder. By specifying a dispatch in the order of Activity 1, Activity 2 the activities are concatenated and the result is displayed in 4.5 (b). Specifically, the Release Side Camera at A is joined with the Claim Side Camera at C. Similarly, the Release Workholder node at B is merged with the Claim Workholder node at D an replaced by a sequence arc. The

edge A+C is transitive so can be removed, because it is made redundant by the dependency B+D. This method for glueing the activities together to form a single DAG for the monolith metamodel is:

- For each resource:
  - Connect its release node to the 'next' claim node.
  - Connect dependencies of the release node to the dependencies of the claim node.
  - Delete those claim/release nodes.
- Remove any transitive edges
- Identify where there are two sync nodes that are directly connected by a single edge. This edge can be deleted and the two sync nodes merged into one.

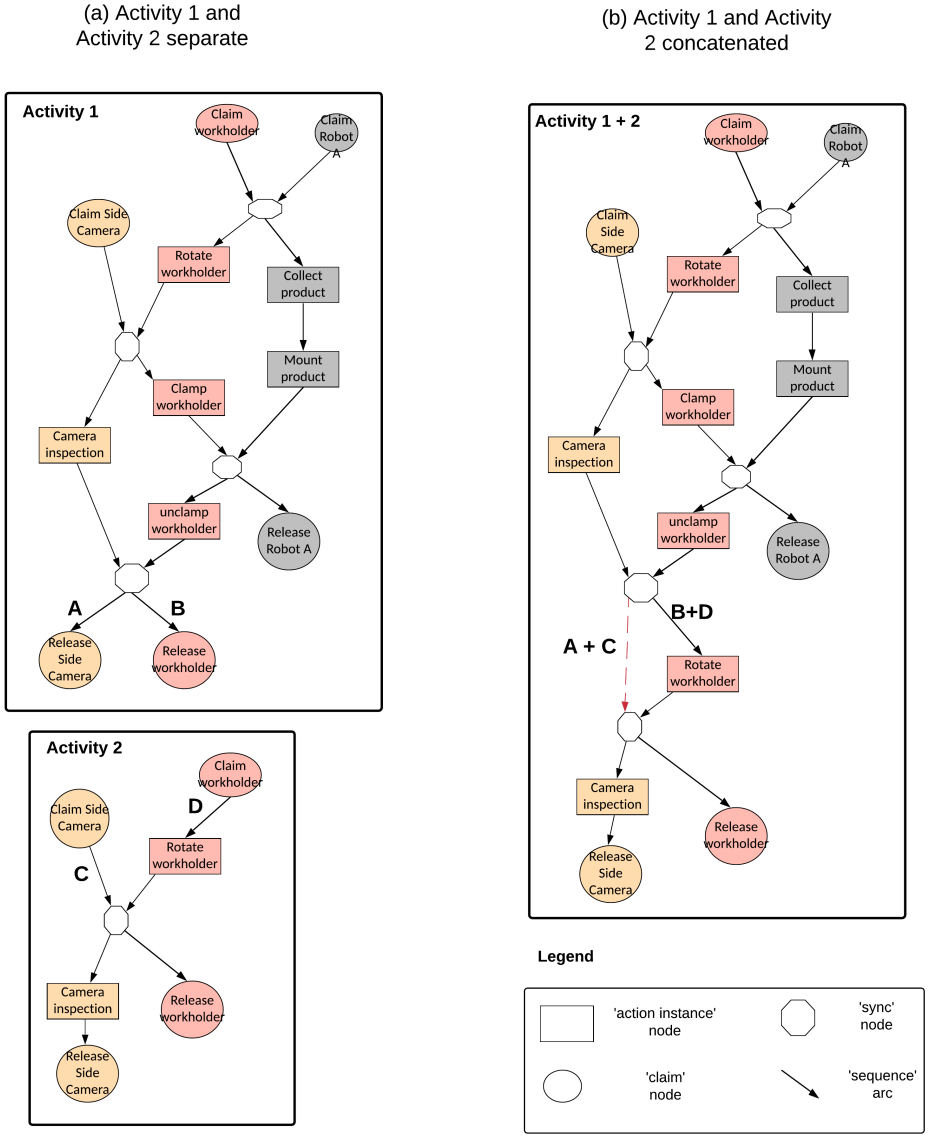


Figure 4.5: Two example activities; separately in (a) and concatenated in (b).

4.2 Decomposition: the identification of components

The purpose of this step in NewMODE is to actually decompose the model and identify potential components within the model, in answer to RQ2:

How can the architectural metamodel be decomposed?

The main requirements for this step is to produce components in an automated manner, such that the model behaviour is correct with respect to the legacy model. This means that the decomposed architecture should account completely for all aspects of the monolith architecture, and could reproduce the behaviour correctly if required. An overview of the goal of the decomposition algorithm is shown in Figure 4.6, where the monolith model is pictured on the left and the resultant decomposition on the right. This decomposition is implemented by analysing the architectural metamodel and applying the algorithm that can identify components from this metamodel. Therefore, the technique must be designed while considering the specification of the metamodel as a directed network.

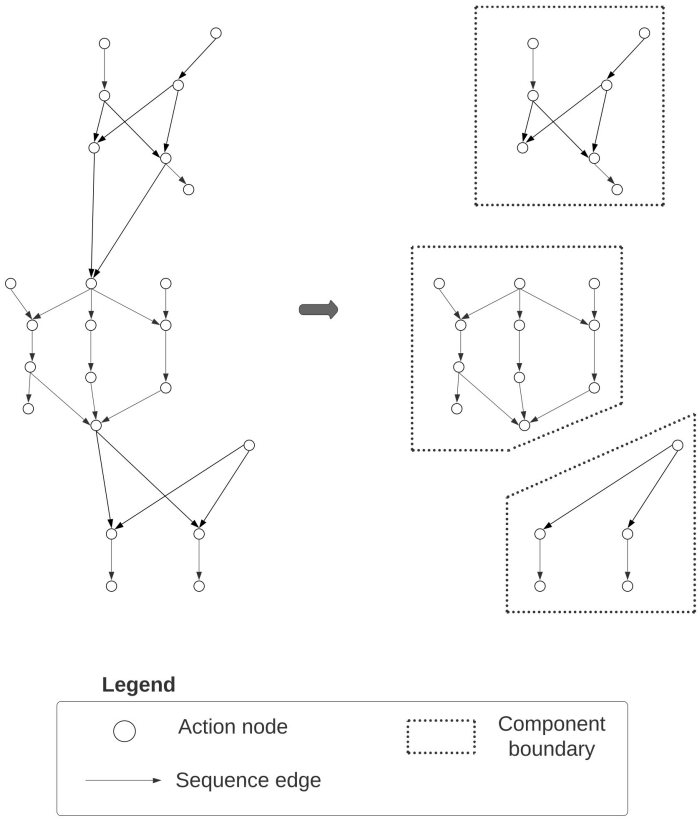


Figure 4.6: Overview of the decomposition algorithm. The metamodel of the monolith (left) is decomposed into three components (right) that are highly cohesive and loosely coupled.

4.2.1 Girvan Newman algorithm

As discussed in Section 2.5, the Girvan Newman algorithm was originally developed for detection of communities in social networks and I will now apply it to model decomposition for NewMODE. Edges are iteratively deleted until the model consists of the desired number of isolated components, as shown in Algorithm 1. Figure 4.6 shows the boundary of components identified from the monolith metamodel. The edges that have been deleted in order to isolate these components will be referred to as boundary edges.

An edge (u,v) is denoted as a **boundary edge** when u and v belongs to different components.

These boundary edges are identified by calculating the betweenness centrality (BC), as shown in Algorithm 2. The BC of an edge is the number of these paths running through it. This can be seen in Figure 4.7 where the edge with the highest BC is shown in red for a trivial case of component detection. When a graph is made of well defined clusters, all shortest paths between nodes in different clusters have to go through the few boundary edges, which therefore have a large BC value. In order to apply this approach to the model, it is important that in calculating the shortest paths between a pair of nodes, the direction of edges is always respected.

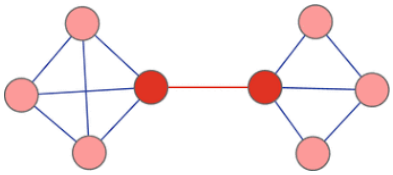


Figure 4.7: A trivial case of edge betweeness centrality: the edge with highest edge BC is highlighted between the dark red nodes. By removing it, two loosely coupled components are isolated.

Algorithm 1 Decomposition

```
1:  procedure DecomposeMetamodel(k)
2:      G ← copy Metamodel
3:      while n <= k do
4:          e ← next_edge_for_deletion (G)
5:          remove_edge (e, G)
6:          boundaryEdges ← append (e, boundaryEdges)
7:          n ← number_of_components_in (G)
8:      end while
9:      return boundaryEdges, G
10: end procedure
```

Algorithm 2 Boundary edges

```
1:  procedure NEXT_EDGE_FOR_DELETION(G)
2:      for all edges in G do
3:          x ← betweenessCentrality (edge)
4:      end for
5:      return edge with highest (x)
6:  end procedure
7:  procedure BETWEENESSCENTRALITY(edge)
8:      for all pairs of nodes (u,v) in G do
9:          p ← shortest path between (u,v)
10:         if edge in p then
11:             x ← x+1
12:         end if
13:     end for
14:     return x
15: end procedure
```


4.2.2 Removing sync nodes

In order to apply this algorithm to the LSAT metamodel there are design choices that need to be made. Firstly we must consider the significance of sync nodes in the metamodel. These nodes represent a dependency between actions in the execution sequence. They can be removed without affecting the meaning of the metamodel if, for every incoming neighbour, an edge is connected to every outgoing neighbour of that sync node. This transformation step is illustrated in 4.8. The main motivation for performing this transformation is that by removing the sync nodes, there is more freedom to the resulting decompositions. For example, in Figure 4.8 (a), components could only be identified by separating actions that are already grouped as input/output to the same sync node, but cannot separate these groups unless all other sync nodes are already partitioned on. In Figure 4.8 (b), where sync nodes are replaces by the edge dependencies, two components can be separated by removing the edges that connect them directly.

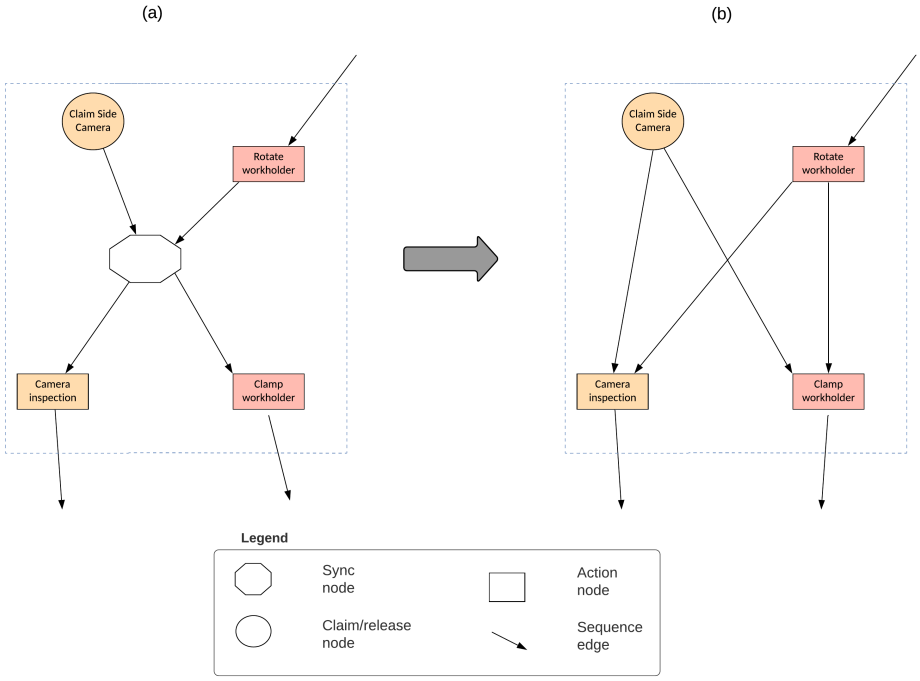


Figure 4.8: Transforming the metamodel to remove sync nodes.

4.2.3 Boundary edges

The decomposition algorithm returns a list of edges that were deleted to retrieve the desired number of components in the system. However, we cannot simply delete these edges from the metamodel if the behaviour of the recomposed system is to be unaltered. As shown in Algorithm 3, the first step to recomposition is to retrieve all the dependencies associated with an edge that has been identified as one for removal.

Next we must check whether the edge is indeed a boundary edge. Considering the intuition of the Girvan Newman algorithm it is possible, although unlikely, that an edge is identified as the next edge for deletion but its nodes end up in the same component. Therefore, the first task in this algorithm checks if this is the case, and re-adds the edge to the metamodel if necessary.

Algorithm 3 Recomposition

```
1:  G ⟵ retrieve_dependencies (boundaryEdges, G)
2:  G ⟵ retrieve_sync_nodes (G)
3:  G ⟵ recomposition_order (G, metamodel)

4:  procedure RETRIEVE_DEPENDENCIES (removedEdges, G)
5:      G ⟵ components_from (G)
6:      for all edges (u,v) in boundaryEdges do
7:          if component (u) = component (v) then
8:              add_edge (u,v) to G                    ▷ Edge is not a boundary edge
9:          else
10:             if resource (u) = resource (v) then
11:                 G ⟵ add_claim_release (u,v)        ▷ Edge is within 1 resource
12:             else
13:                 G ⟵ add_ghost_resource (u,v)       ▷ Edge is across 2 resources
14:             end if
15:         end if
16:     end for
17:     return G
18: end procedure

19: procedure RETRIEVE_SYNC_NODES( (G))
20:     for all components x in G do
21:         if node has inDegree < 1 then
22:             G ⟵ add_sync_node (node)
23:         end if
24:     end for
25:     return G
26: end procedure

27: procedure RECOMPOSITION_ORDER( G, Metamodel)
28:     branching ⟵ branching_forest_from (G)
29:     for all components x in G do
30:         r ⟵ roots (x)
31:         ranking(x) ⟵ level_in_tree (branching, r)
32:     end for
33:     return ranking
34: end procedure
```

Otherwise, it is a boundary edge (from node u to node v) and there are two cases that must be considered when dealing with this which can both be seen in Figure 4.9 [1]. In this illustration action nodes are coloured by the resource that executes the action, according to the same colour scheme used since Chapter 3. On the left hand side of this figure the 4 boundary edges are shown; A, B, C, D. The first case of boundary edges is straightforward and occurs when both u and v belong to the same resource - edges A and C in the example. By adding an edge from u to a newly created release node for this resource, and adding an edge from a newly created claim node to node v, the correct dependency between components is maintained. This can be seen in the transformed edges on the right side of Figure 4.9.

Case 1: a boundary edge (u,v) where u and v are actions that are executed by the same resource in the machine.

4.2.4 Ghost resources

The solution to the second case of boundary edges is less straightforward. This issue arises when a boundary edge starts in one resource and ends in another, like edges B and D in Figure 4.9.

Case 2: a boundary edge (u,v) where u and v are actions that are executed by different resources in the machine.

To tackle this I introduce the concept of a *ghost resource* to force inter-component synchronisation, following a similar intuition to Dijkstra's semaphore [50]. A ghost resource does not represent a physical resource in the machine, and does not have any actions, but is used to force a synchronisation between components, when a boundary edge crosses two resources. This is done by creating a 'claim ghost' $\longrightarrow$ 'release ghost' sequence for each component, and adding an edge from u to the claim ghost in its component, and v to the claim ghost in v's components. Because LSAT knows that only one activity can occupy a resource at a time, the two components will be forced to wait at this ghost resource, thus imitating a synchronisation point across the components.

[1] Figure 4.9 shows the boundary edges that result from the decomposition used as an example in Figure 4.6.

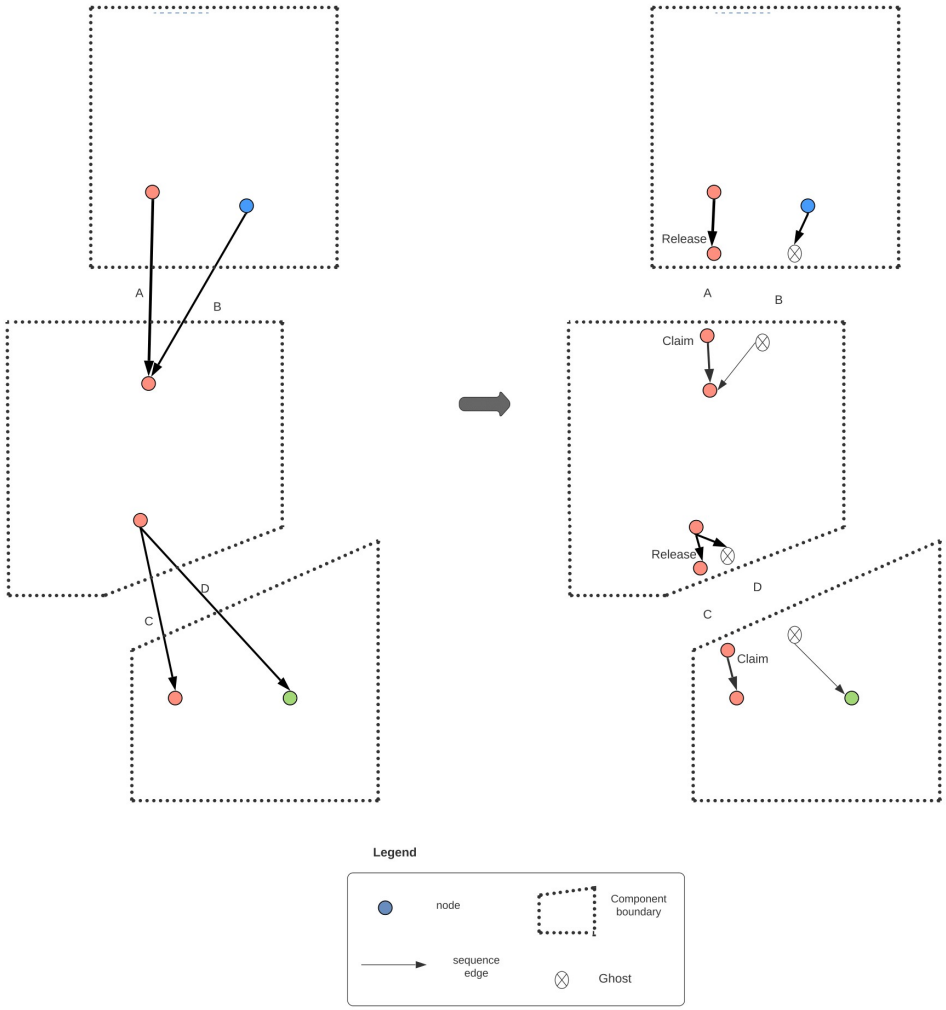


Figure 4.9: Boundary edges A, B, C and D are transformed to maintain synchronisation correctness.

4.2.5 Recomposition order

The final step in recomposition is to identify the order in which the components should be executed. This is done by converting the original, monolith model into a form that shows the order of each node as a *branching*. This representation is a directed forest that has all (overlapping) paths from root nodes to leaf node in the given directed, acyclic graph. The paths are such that any node B that is dependent on a node A is shown in a level lower than A in the branching. The idea is to identify the root of each component and then search for its position on the monolith branching. Then, this can be used to construct the order of execution of these components in a distributed architecture. Due to the way analysis is performed in LSAT, any components that start on the same level in the branching will be

performed in concurrently in LSAT, so their order doesn't matter. Otherwise, the order follows the ranking of the components according to the position of their root node in the branching.

4.3 Criteria: evaluate the decomposition

Decomposition criteria are the quality metrics evaluated for a given model architecture. They represent the quantitative evaluation of how (un)desirable a given model architecture is and relate to RQ3 in this thesis.

RQ3: How can model decompositions be evaluated quantitatively?

Based on the literature discussed in Section 2.1, the first assumption is that decomposition criteria depend on the driving forces for decomposition, as proposed by Hofmann [26]. Therefore the design choices for this section depend heavily on the use case and the following observations:

- The ideal decomposition architecture minimizes the size of components and the number of components in the system. This paradox is referred to as the granularity trade off in literature [21].
- The model is used for the design phase of the Digital Twin. Therefore, interpretability of components is desired.
- Reusability of model components is desired.
- Performance of the model is not a main priority.

Furthermore, the decomposition architecture is evaluated on separate levels: the individual component level and the overall system level. It is assumed that components are not uniform in size, cohesion, coupling etc. Therefore the variance of criteria on the component level need to be accounted for when aggregating to the system level.

4.3.1 Internal complexity

The internal complexity of a component is minimized in order to achieve high cohesion and interpretability of components. It also contributes to the maintainability of components and their re-use potential. The first criteria for internal complexity is size. Specifically, *size of a component* is calculated as the number of action instances in each activity in LSAT. The action instance is the base unit in LSAT and so its count gives an indication of size. However, not all components of the equal size are equally complex. For example, a string of linearly sequentially actions may have a large size but the linearity makes it more comprehensible than a component of the same size with many internal connections. Therefore we introduce a second criteria for internal complexity that is referred to as *link density*. Specifically, link density is calculated by dividing the number of internal edges (links) in a component, by the size of the component. A link density of exactly 1 means there is the equivalent of 1 incoming edge for each action instance node, therefore there is less synchronisation across resources and the component is very linear in its description.

4.3.2 External complexity

Within social network science we find a definition of community strength [45]. This leads us to calculate the external connections compared to the size of a component[2]. A universally understood trade-off when migrating to a distributed model architecture is the effort involved in defining the interface that connects all the components, a task that is unnecessary in a monolith system. The number of inter-component synchronisation points outgoing from a component is an indication of the effort involved in building such an interface layer. Therefore, the number of ghost nodes in a component is an important criteria for decomposition. To account for the size of components, we define *ghost density* as the number of ghosts divided by the size of a component. This can be calculated for each component in a candidate and should be minimized by the ideal decomposition.

4.3.3 Criteria ranges

As shown in Table 4.1, three criteria for evaluating *individual components* were discussed with the model owner and experts to describe an expected range for these criteria based on domain experience. The intuition behind selecting link density and ghost density is that they account for the trade off between internal complexity and external interface complexity. Minimizing the size of components is desired as a simplistic measure of internal component complexity. This directly conflicts with the number of components in the candidate decomposition. By definition, the mean size of a component decreases exponentially with the number of components in the system but the variance in size of components for a single candidate architecture is not constant. This means that candidates might have 30 components that vary in size from 7 to 80, for example. Therefore, it is important to consider the variance in size of components, especially to identify extremely large values. Even for two components of identical size, one may be more desirable than the other based on the second and third criterion in Table 4.1.

Table 4.1: Criteria for evaluating individual components.

Criteria	Acceptable range
Size	5 - 20
Link density	1 - 3
Ghost density	< 0.5

A common challenge in manual decomposition is deciding the number of components to design in the system. *The level of granularity* is therefore an important system criteria that is calculated as the number of components in the candidate decomposition. When hierarchically decomposing a system the stopping condition for granularity refers to some rule of guideline to stop partitioning further [21].

4.4 Compare candidate decompositions

Thus far, each of the three research questions have been addressed and resulted in the specification of three steps in NewMODE; metamodelling, decomposition, criteria. To-

[2]Parisi et al. used this slightly differently to make a binary definition of strong or weak communities based on this ratio less than or greater than 1.

gether, these steps form a process that can decompose a model from its code into a set of components, **for a specific number of components** in the system. In the state-of-art in model decomposition (Section 2), the number of components in a decomposition is typically left to user-input and expert preference. For example, in Mazlami's algorithmic approach to microservitization he remarks:

> *".. it is very difficult to determine when to stop partitioning the graph. As a consequence, the algorithm must take the number of targeted partitions as an input parameter."*

A similar argument is given in [32] and [6] that the user must input the number of components desired. More importantly, the application of the GN algorithm for model decomposition also relies on a desired level of granularity.

Considering the importance of the number of components in the decomposition architecture and the lack of guidance in literature for selecting this number, I have decided to take a novel approach to tackling the granularity trade-off. Therefore, the last step in NewMODE is added in order to guide developers in selecting a candidate decomposition with an optimal level of granularity, instead of relying on the user to determine it. Iteratively decomposing the metamodel for different levels of granularity results in a set of candidate decomposition solutions that can be compared using the decomposition criteria. I consider this approach to be an important contribution of this thesis to the state-of-the-art. This iteration is illustrated in Figure 4.1 and follows a straightforward algorithm, shown in Algorithm 4. This results in a dataset of n candidate decompositions, that can then be compared using the decomposition criteria.

Algorithm 4 Explore granularity

```
1: procedure MyProcedure
2:     monolith ⟵ concatenated metamodel
3:     k ⟵ sample numComponents
4:     for all n do
5:         decomposition ⟵ decompose monolith to size k
6:         x ⟵ evaluate decomposition
7:     end for
8: end procedure
```

4.5 Implementation

The implementation of NewMODE has been built with flexibility in mind such that the design aspects that are most likely to be changed are self-contained. For example, the LSAT-connect.py module can be replaced to integrate with additional modelling languages other than LSAT. Similarly, the decomposition.my_girvan_newman function can be changed to test out additional decomposition algorithms. The candidates are output in a standard file format for graph exchange (.gexf) to allow users to choose their preferred visualisation tool. NewMODE is implemented as a Python package with four modules:

- **LSATconnect.py**: this module converts model code written in LSAT into NewMODE's metamodel structure.
- **Metamodel.py**: contains the definition of the Metamodel class; its attributes and functions. The Metamodel class is at the core of NewMODE and extends from the networkx.MultiDiGraph class for multi-edges, directed graphs.
- **Decomposition.py:** this module implements the decomposition algorithm as described in Section 4.2.
- **Analysis.py:** contains functions to evaluate the criteria for decomposition candidates and produce visualisation of results.

NewMODE outputs to a single results directory with the following files:

- **SystemCriteria.csv:** a dataframe with one row for each candidate decomposition.
- **ComponentCriteria.csv:** a dataframe with one row for each component, in each candidate decomposition.
- **Candidates:** This is a folder that contains one file for each of the candidates. These files are in the Graph Exchange XML Format (.gexf) which can be viewed in Gephi, a software for network visualisation.

Appendix C.1 lists the dependencies on which NewMODE relies and Appendix C.2 provides example code of how to use NewMODE for a simple example, including where to use the metamodel class and the various modules. For access to the source code please contact l.crowley@student.tudelft.nl.

4.6 Summary

This Chapter relates directly to the design of the proposed methodology and answers the research questions in Chapter 1. The architectural metamodel can represent the monolith system as a directed network. It is possible to capture the dynamic dependencies between action instances as well as their static class dependencies using layers in the network. In response to RQ2, model decomposition can be automated, given the specified metamodel. By applying a series of horizontal transformations on this metamodel, candidate decompositions are produced that consist in a number of components with well defined boundary and clear interfaces. This is achieved by iteratively removing edges from the network until components are isolated [47]. Edges are removed based on their value of edge betweeness centrality and then become boundary edges that connect between components. The proposed decomposition is iterated to find a set of candidate decomposition architectures of different levels of granularity, where granularity refers to the number of components in the system [6]. Finally, RQ3 is addressed with decomposition criteria that evaluate the quality of identified components. The definition of this criteria depends heavily on the driving forces for decomposition and the topology of the system under decomposition.

5

Evaluating NewMODE

The following chapter describes the evaluation of the proposed NewMODE methodology by comparing its results to a manually decomposed model in LSAT based on the criteria in Table 4.1. In Section 5.1, the reference models that are used for comparison are briefly introduced. In Section 5.2, the automated decomposition is compared to the reference models using the quantitative decomposition criteria. In Section 5.3, the results of the face evaluation with the expert are discussed.

5.1 Reference models

In many academic domains there exist standard test models or topologies that are accepted as good testbeds for scientific research. For example, in network science there are standard network topologies that represent the desired or undesired characteristics on which new algorithms can be evaluated. Similar standard testbeds exist for many domains but this is not the case for model decomposition. Considering there is little research into model decomposition this is not surprising. Therefore, the methodology can only be evaluated for a pair of monolith and modular models built in LSAT by experts in the field.

The proposed decomposition methodology was implemented for a working model built in LSAT of a high tech manufacturing process. This model was built by scientists in TNO's ESI group in consultation with one of their partners in the manufacturing industry. Scientists in ESI built the model in the *monolith-first* approach, whereby the model was developed as the original (monolith model A) and then later decomposed into a component based model (modular model B) during development. The developer of this model is an established expert in model based systems engineering with dozens of publications in this domain and his evaluation of the decomposition is considered as an expert evaluation of the proposed decomposition methodology. In order to evaluate NewMODE, I compare the results of the automated decomposition, C, to both the original model, A, and the manual decomposition, B, using the criteria developed in Section 4.3. Although the monolith and modular models are extremely similar in that they represent the same physical machine, the modular version represents a slightly different version that is implemented for a different dispatch sequence, and thus a smaller model overall. It can be assumed that the topology of the decomposed model represents the desired architecture of its monolith

model, although we must be careful to remember the difference in overall size of these models when comparing the number of components and size of the model. The monolith has a total size of 718 action instances, while the decomposed model has a total size of 267 action instances.

Table 5.1 summarises the manually constructed models according to the criteria, which led to an unexpected finding in itself. Notice that the average size of components in B is just below the lower limit of the expected range cited by the expert, which verifies the magnitude of this range. Equally, the average size of the monolith is, as expected, larger than the desired range. It is unusual that the link density of the decomposed model is 4.24. This indicates that there are 4.24 internal edges in a component, for every 1 action instance in that component. This suggests a high number of sync nodes or claim and release nodes compared to the number of action instances. In essence, this suggests that the internal complexity of the components are reasonable high, which gave reason to visually inspect the model code of the components with high link density. In fact, this uncovered a number of redundant nodes and edges within these components that presumably got left in a component that was manually decomposed. For example, the claim and release node of a resource are connected to a sync node, despite that resource never being used in the component. This results in unnecessary duplication of code when manually decomposing components. Therefore, the first unexpected result of this thesis is that it can be useful in analysing decompositions that have been completed manually, even identifying unnecessary code duplication and internal complexity, a tedious task to verify manually.

Table 5.1: Comparing the characteristics of the manually developed models: A is the monolith model and B is a version of A that has been manually decomposed.

		Component criteria				
Model	Absolute size	Size	Links	Link density	Ghost	Ghost density
A: Monolith	718	22.44	72.89	3.19	-	-
B: Decomposed	267	4.45	18.85	4.24	1.01	0.23

5.2 Quantitative evaluation

The monolith model was decomposed using the proposed decomposition methodology across a wide range of granularity. The model was decomposed from 2 components up to 140 components, therefore resulting in 139 different possible decomposition architectures that each represent a correct representation of the Monolith model A at a different level of granularity. The upper limit of 140 components was chosen as the algorithm began to identify candidates where the average size is less than 2 action instances per component, and many components of size 0. As shown in Figure 5.1, the candidates C can be compared to the monolith A and the manually decomposed model B.

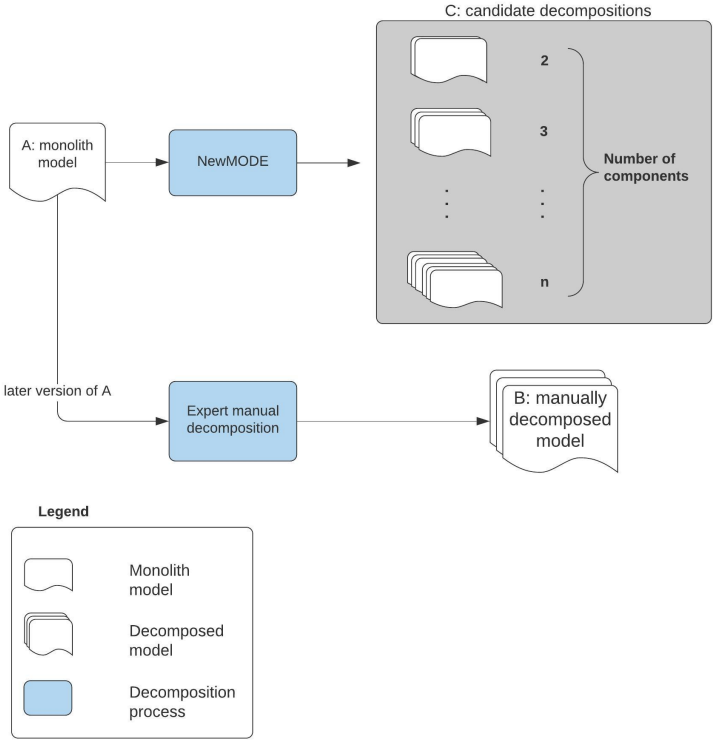


Figure 5.1: Overview of the models used for comparison.

5.2.1 The Pareto front

Considering the decomposition method results in a large number of candidates, it is difficult to compare them according to the multiple objectives. The first filter in identifying optimal components is to determine the Pareto optimal front which results in set of candidates whereby the performance on one objective cannot be improved on without worsening the performance on another objective [51]. To implement this Pareto dominance, we must be able to directly compare the candidates with the criteria. Considering the criteria above are recorded for each component in each candidate, it is necessary to aggregate the results of each component for a single candidate. Of course, the simple choice for aggregation is the mean, but this does not account for the variance in criteria across different components within a candidate. As an example, the variance of component size in a candidate can lead to components of size 30 and others of size 2 which is undesirable for model management. Instead it was decided to use the product of the mean and the standard deviation to aggregate the criteria for a single candidate. The standard deviation is an appropriate measure of the variance in this case, as for each individual candidate, the value of each criteria approximately follows a normal distribution. Considering we want to minimize both the mean and the variance for all criteria, this product is an appropriate

choice [52]. This resulted in 19 candidates in the Pareto set at various levels of granularity. Although these candidates are not discussed separately, they provide a modeller with the choice to filter further by their preferred criteria or hard constraints.

In the following sections, the candidates will be compared across the entire range of granularity for each criteria in turn starting with internal complexity and finishing with external interface complexity. The perfect candidate minimizes all criteria, including the level of granularity.

5.2.2 Internal complexity

Size of components

First, we can explore the size of components in Figure 5.2. The x-axis is the level of granularity whereby there is one candidate for each granularity value. For a single candidate, you can see the distribution of the size of each of its components as dots plotted vertically at that level of granularity. Alternatively, it is possible to see the relationship between median size and granularity of components by following the black line. Additionally, there are two reference lines included in this plot. The blue horizontal line represents the value of this criteria for the monolith model A. The red horizontal line represents the value of this criteria for the modular model B. Before granularity of 20 the size of components is considerably larger than the expected value of 5 - 20. After 100, the candidates have components with 0 size which indicates that further decomposition is not valid for this algorithm. Therefore, Figure 5.3 shows a subset of this graph for levels 20 to 100 to get a clearer image of the behaviour. It seems there is a discreteness to candidates when considering their median size. Comparing this to the mean size of components in the monolith model A and the decomposed model B, it is seen that after decomposing the model into 20 components it performs similar to the monolith model A. Similarly, the candidates perform similar to the modular model B after 80 levels of granularity.

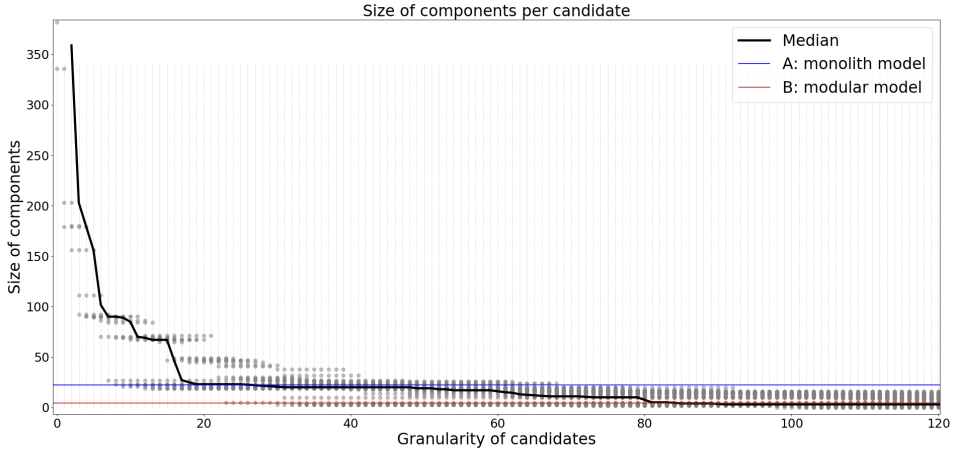


Figure 5.2: Comparing candidates based on component size.

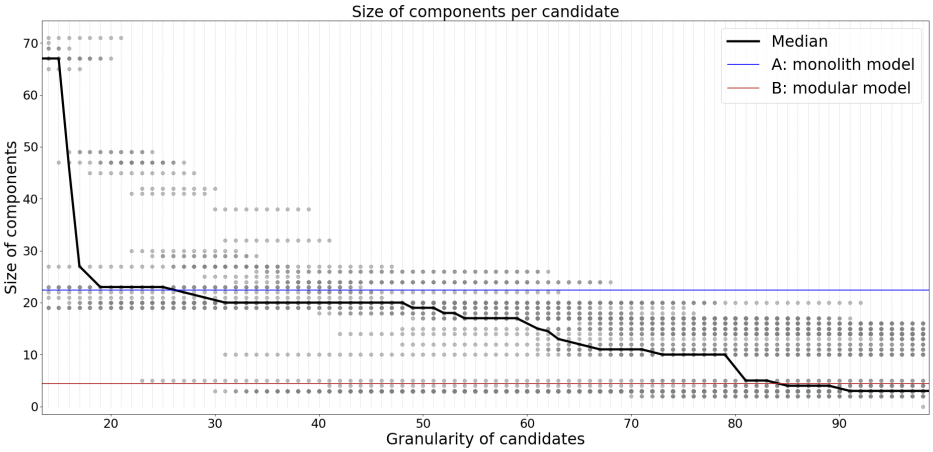


Figure 5.3: Comparing the distribution of component sizes for up to 100 levels of granularity.

5

Link density of components

Next, we can inspect the internal link density of components in Figure 5.4. In the Appendix D.2, and Appendix D.3 the mean link density is plotted against granularity as boxplots. An internal link density of 1 means that on average, every component is dependent on one other in the execution sequence. Therefore there is little synchronisation between different resources within a single component and the activity of this component can be considered relatively *linear* to comprehend, even if it has a large size. For values greater than 1, each action instance is dependent on more than one other and there are many sync nodes within the activity. As shown by the reference lines, the automated decomposition consistently finds components that perform well in this criteria. The relationship between link density and size of components is noticeably discrete. By following the median of link density it is possible to distinguish a step wise decrease in the mean link density around 82 to 86 levels of granularity. Remembering that increasing the number of components is generally undesirable and the same applies to the link density, it would not be advised to decompose past 90 levels of granularity based on this criteria alone. In a similar reasoning, it would be ill-advised to decompose the model to 79 levels of granularity as this would have a similar level of mean link density as having 48 components. This plot is also useful in identifying particularly internally complex components so that one can avoid decomposition architectures with a handful of aggressively internally complex components.

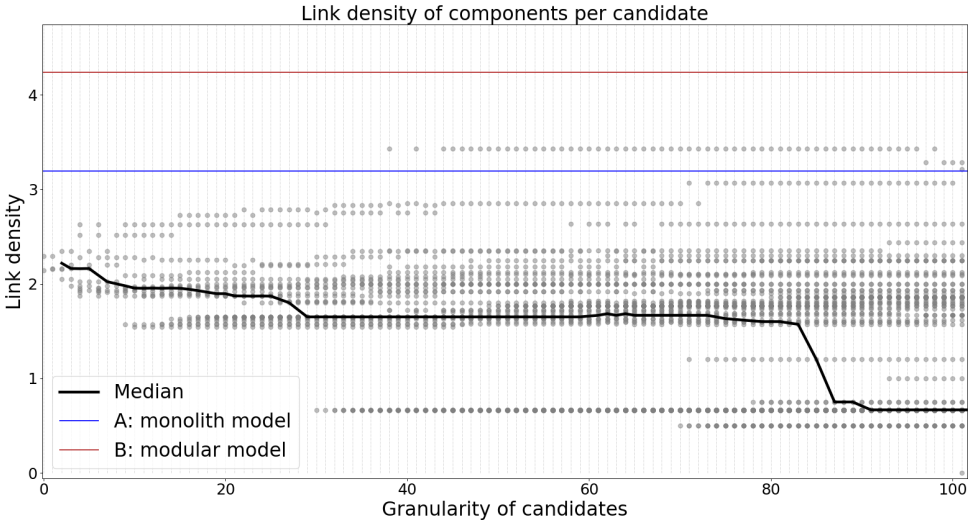


Figure 5.4: Comparing the distribution of internal links in each component.

5.2.3 External complexity

Ghost density of components

The first conclusion is that the automated decomposition results in components that are larger in size alone, but less complex in terms of internal interactions. This leads to the concern that the external complexity will be large as a result, which would be undesirable. Therefore, we examine the number of ghosts per components; ghost density.

As shown in Figure 5.5, the distribution of ghosts has a strong left skew for granularity above 30. Specifically, most of the components in these candidates have 0 ghosts but outliers have 1 ghost for every 3 or 4 ghosts action instances in that component. Even these outlying high values are within the acceptable range for ghost density. To verify this pattern of ghost density we can also examine the absolute number of ghosts in each component in Appendix D.6 to provide additional insight. The number of ghosts is exponential (see Appendix D.4) so is displayed with a log scale. The number of ghosts in a component follows an interesting pattern. Up until a threshold level of granularity, the number of ghosts is similar for each component at a given level. After crossing this threshold there appears to be a many components with 0 ghosts and other components with a relatively large number of ghosts. This might be an indication of a **stopping condition** for decomposing past this threshold level of granularity, given that the number of ghosts is a proxy for the complexity of the external interface each component has.

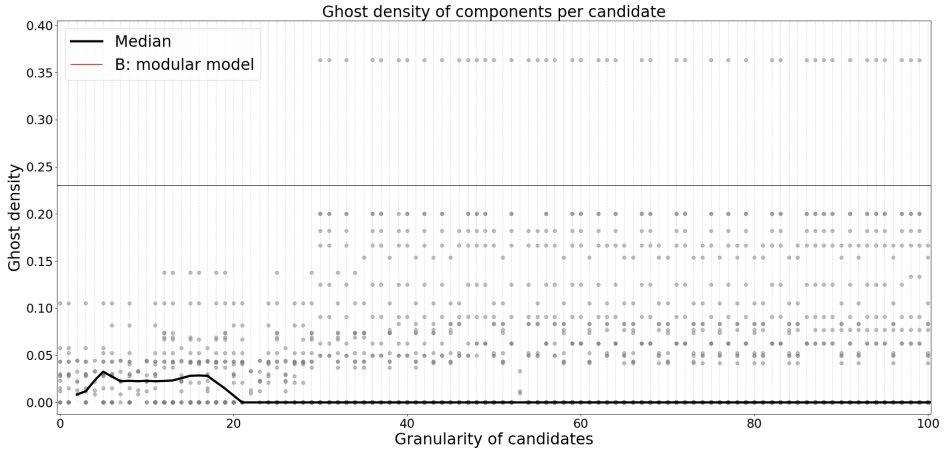


Figure 5.5: Comparing the ghost density in each component, across candidate decompositions.

5

5.3 Expert evaluation

In order to verify the intuition of the model decomposition process and gain additional insight, a face evaluation was conducted with the expert and developer of the LSAT model. This involved a presentation of NewMODE as outlined in Chapter 4 followed by graphs of the criteria for the candidate decompositions.

The overall response to NewMODE was that it felt similar to the logic the expert follows when manually decomposing a model in LSAT. As explained in Section 4.1.3 one of the rules for the metamodel is to translate dispatch sequences into the network. This was done by concatenating the relevant Activities and merging the closest claim and release nodes for each resource. An expert who was involved in the conceptualisation and development of LSAT, stated that this design choice seems correct because it imitates the behaviour that LSAT's analysis is completing 'under the hood'. A similar validation was given for the choice to remove sync nodes and replace them with dependency edges, as described in Section 4.2.2.

5.4 Summary

Table 5.2 shows the results of the automated decomposition, along these three criteria, in comparison to both the monolith A and decomposed model B. For example, the candidates that have component size that is within the expected range from the expert is shown in the top left field as levels 20 - 80. This shows that the proposed candidates are always better than the monolith A and modular model B in terms of link density and ghost density. For some candidates, they also perform better that A and B for component size. In conclusion, the automated decomposition improved the internal complexity of components, and the levels of external complexity (ghost density). The size of components is only comparable to the modular decomposed model after 80 levels of granularity. In general, the automated decomposition identified components with a higher variation in size, for the same mean

Table 5.2: Comparing the candidates C to monolith A and modular model B.

	Granularity level where C is...		
Criteria	**...in expert's expected range**	**..comparable to A**	**...comparable to B**
Size	20 - 80	20 - 50	80 - 100
Link density	2 - 100	2 - 100	2 - 100
Ghost density	2 - 100	-	2 - 100

size of components, when compared to the expert decomposition model B. This is largely due to the intuition of the algorithm itself. Additionally, these larger components appear to be relatively linear or internally straightforward, in that most action instances within a component were dependent on approximately 1 other action instance in the same resource. The Pareto front can be used to create a shortlist of candidates for further selection based on further constraints or preferences.

6 Discussion

6.1 Limitations

It is difficult to recommend a single decomposition architecture from the set of candidates as this depends on the aggregation of multiple objectives. The Pareto front is useful in providing a shortlist of these candidates but due to the difficulty in visualisation of the candidates, it is difficult to choose between them confidently. It would be possible to apply a utility function to the multiple objectives, possibly with a weighting of the relative importance of different criteria, but this would only be advised when there is stronger trust in the criteria themselves.

One limitation of NewMODE is that the decomposition can result in a number of large components in a candidate set, which are outliers in terms of their size. It is not yet clear whether this is desirable. Equally, it may be a result of the decomposition algorithm or the inherent topology of the system itself. Another limitation of NewMODE is that it does not identify components that are repetitions of another component. Ideally, the components identified should be checked to identify repeated components, as they definitely exist based on a visual inspection.

This study was compared to two models in the LSAT formalism and needs further analysis on models within this class to generalise the results. The lack of standard testbed or repository of distributed model architecture limits the results of this study. The availability of pairs of models in monolith and modular form would allow more robust scientific research into the effectiveness of model decomposition methodology, both for testing the effectiveness of proposed methodology and for analysing the characteristics of good decomposed architectures.

6.2 Reflection

The aim of this section is to spark discussion and reflect on various aspects of this thesis.

6.2.1 The Girvan Newman algorithm

The decision to use the GN algorithm adds flexibility and generalisability to NewMODE. This algorithm lends itself to customisation for a specific type of model, domain or driving

force for decomposition beyond what is implemented in this thesis. When selecting the next edge to cut, NewMODE currently uses the edge betweeness centrality but other criteria could be considered and easily replace this function in the implementation. For instance, rules can be added to this function to define a minimum size of the component being split or the number of resources being used within a component. Another possible customisation would be a stopping criterion based on criteria for the metamodel.

Another reflection on this algorithm is that further exploration of candidates can be completed by introducing stochastics to the rules for identifying components. By introducing randomness, the algorithm will find more than one candidate decomposition for each level of granularity. This could be implemented simply by adding some noise to the edge betweeness centrality calculated for each edge in the network. The benefit of adding stochastics is that further exploration of the ideal candidates is likely to provide insight into the optimal method for decomposition. Furthermore, this opens the possibility to use NewMODE to implement a multi-objective optimisation to search for optimal candidates.

Each step of the proposed algorithm involves a cubic algorithm ($O(n^3)$), which is a threat for scalability and the ability to apply NewMODE to extremely large models, especially if employing a non-deterministic adaptation as discussed above. One option to decrease the time of the algorithm involves hierarchical decomposition whereby the monolith model is first cut into two equal-size parts, which then are decomposed individually. Then the next steps become much cheaper than the first steps. Of course, the results will depend heavily on this first initial cut which might limit the value of the final candidates.

Visual validation of candidate decompositions could be used to define specific customisations of the algorithm on a case-by-case basis. Equally, this would be useful to explore the architecture of manually decomposed models. Considering NewMODE outputs the models in the standard Graph Exchange XML Format (GEXF), the model can be visualised using any preferred software[1].

6.2.2 Ghost nodes

A key finding in this thesis is the automated identification of *ghost resources* to implement inter-component synchronisation. Recall from Section 4.2.4 that they are needed when a boundary edge is identified that crosses between two different resources. Logically, this is due to the fact that sync nodes are used in LSAT when there is synchronisation between two resources in the same activity. After discussing this with the expert, he explained that these ghost resources are an interesting design choice in NEwMODE. Typically, when manually decomposing a model, they find that these synchronisation nodes are used when there is a physical requirement, based on domain knowledge from the physical process. For example, knowing that Robot A can't complete an action while Robot B is in a certain state is derived from knowing the physical process itself.

Using NewMODE, it is possible to identify these 'ghost resources' algorithmically, without any prior domain knowledge. The benefits of this are manifold. Firstly, there is the time saved when doing this with NewMODE versus with domain knowledge. Secondly, the formalisation of these ghost resources has advantages for knowledge transfer in the

[1] I recommend Gephi as it has convenient layout options, is open source, and display settings can be saved and applied to graphs automatically.

domain of modelling. According to the expert, this inter-component synchronisation is a 'handy trick' which, when formalised, can be easily spread and adopted by more people.

Readers who are familiar with Petri nets will see the similarity of the ghost concept to the lock in this formalism. Upon reflection it is clear that the second layer of NewMODE's metamodel bares resemblance to the core concepts in Petri nets for modelling of concurrency and deadlock in control systems [53]. It would be interesting to explore the possibilities of specifying the second layer of the metamodel as a Petri net where each physical resource uses tokens and locks can model ghost nodes.

6.2.3 Metamodelling formalism

In fact, the similarity of ghost nodes to Petri net's lock concept leads to a deeper reflection on the use of network theory as the metamodelling formalism in general and what formal methods are relevant. This metamodel was built with three layers, the first of which describes static dependencies in the model. Many concepts from this layer are borrowed from UML and in fact the entire layer could be conceived as a UML model. As discussed the second layer represents the execution order of model actions which could be modelled as a Petri net. For example, Qinyi et. al have demonstrated a transformation of a DAG to Petri net for simulation of concurrent systems and synchronisation between DAG components [54]. Finally, the third layer represents the decomposition architecture which is truest in form to a typical network model.

The network theory approach for metamodelling was originally chosen for the flexibility and scalability it offered over the other approaches investigated. In fact, Petri nets and UML are modelling languages based on graph-grammar themselves and so they could be embedded in a metamodel that is based on graph theory also. By choosing the base formalism as network modelling it has provided the flexibility to explore the requirements of the metamodel and has not restricted NewMODE's design. Considering the other aspects of NewMODE are appropriate for a graph based model (the GN algorithm for example), it is conceivable that these design aspects would still be suitable if Petri nets or UML were used in the metamodel.

Another reflection on the metamodelling formalism is that there is considerable excess work needed to express the given model in a metamodel before decomposing it. This means that developers must be familiar with the metamodel as well as the model formalism itself. Specifically for this implementation of NewMODE, developers who work with LSAT need to familiarise themselves with network theory to understand the metamodel and inner workings of the decomposition. An elegant solution to remove this excess work is to use a *self-reflexive* approach, whereby the metamodel can be described in the same language as the model. According to the example used in this thesis, this requires building the metamodel in LSAT itself. If NewMODE were implemented for decomposition of Petri net models, it is interesting to consider the metamodel constructed as a Peri net itself.

6.3 Impact

Responsible research must consider its impact in the context of all stakeholders, organisations and society at wide. This section reflects on the impacts of this thesis topic on model developers, the organisations they are part of, and society at wide.

6.3.1 Model developers

Formalising methodology

Formalisation of modelling techniques, such as model decomposition, is challenging but rewarding. In one sense it moves towards an automation of the processes which are tedious and time-consuming for modellers by providing them with tools that aid a better decomposition design. Aside from automation, formalising modelling techniques also allows the manual process to be clearly defined. This has practical implications for ease of knowledge transfer in the modelling domain, both in academics and industry.

Model management

One of the challenges that ESI encounter is *model management.* As they partner with many organisations in industry, they see an increasing number of teams who struggle to manage the complexity of their models. As their manufacturing processes grow in complexity, model based systems engineering has been developed as a solution to manage the complexity of these manufacturing process. Models allow them to optimize, monitor and control these complex processes. Of course, this has led to an increasing complexity in the models themselves, at a rate that is higher than the development of modelling skills within industry or scientific consultants like ESI themselves. Automating tasks like model decomposition is extremely valuable as more and more organisations adopt technologies like Digital Twin. There is value in using an approach like NewMODE to aid developers in identifying a starting point for model decomposition, even if they have to make manual adjustments as well.

6.3.2 Organisations

Breaking down silos

For large scale models within an organisation, the structure of the submodels co-evolves with the structure of the organisation's development teams. Effective model decomposition can influence the structure of an organisation and determine the success of model management within that organisation. The realisation of Digital Twin technology within organisations relies on a transdisciplanary perspective, whereby simulation environment is domain agnostic and crosses traditional functional boundaries within an organisation [55]. This represents a shift from traditional organisational silos whereby simulation technology is domain specific and resides with the relevant development team. Digital Twin requires a simulation environment whereby functional teams build components in their domain specific modelling language and the components are co-simulated from a global environment. Modular architectures in these large scale models can influence the structure of organisations and break down traditional silos.

Informed decision-making

In many ways, Digital Twin is characterised as a technology that uses data to inform decision-making of the real world. It allows engineers to explore the state space in real time and during the design phase, instead of taking worst case scenarios or average estimates of known unknowns. Real time control can be informed by scenario discovery and front-running simulation [55]. Decision-making becomes more informed when an organisation can simulate future scenarios, embracing uncertainty in the exploration space and not being

limited to static control. Decomposed simulations are explicitly linked to this possibility whereby multiple instances of the same simulation can be instantiated, sometimes referred to as 'Digital Sisters'. In many ways, simulation of physical systems is shifting organisations away from a traditional 'over-engineering' design approach to more proactive and efficient monitoring and control.

6.3.3 Society

Widespread application of Digital Twin

NewMODE was developed with the aim of operationalising Digital Twin technology. In this thesis it has been demonstrated for a use case from the logistics and manufacturing domain but there are many other applications of Digital Twin, for which the same challenges in modularity apply [56] . For example, the benefits of Digital Twin can be demonstrated with the case of smart grid operations; the uncertainty in human behaviour and it's impact on grid demand can be tackled with Digital Sisters and scenario discovery leading to real-time control of power supply [57].

The sustainable development of our future society rely heavily on digitalistion. This includes but is not limited to the path to decarbonisation with smart grid technology [58], the future of mobility with intelligent traffic systems , and improving quality of life with smart health technology [59]. At a governance level, the appropriate use of higher level Digital Twins can increase the effectiveness and efficiency of decision making across different policy domains.

Cyber-physical systems

Digitalisation brings technology closer into peoples lives. In this way, the gap between technology and society is closing and Digital Twin is expected to have widespread affects on the way we live our lives, from the workplace to the home and beyond [9]. Now more than ever it is important to embed the values of society within the values of technology like Digital Twin, considering it will interact directly with our lives. Autonomous control and decision making should reflect the values of the society and culture in which they are operating bearing in mind that this may also vary across cultural boundaries.

Digital Twin relies heavily on the mutual advancement of IoT and 5G networks which are only as strong as our cyber-security mechanisms. Considering Digital Twin in society means additional resilience is essential to protect our systems from fault, downtime and external hacking. In addition, increased legislation related to digitalisation is necessary to advance our digital society, whilst encouraging industry innovation.

6.4 Summary

The limitations of NewMODE are related to the size of components and comparing candidates for a given model. Furthermore, the thesis is limited by the number of models used for evaluation. This thesis sparks many reflections, due to its novel nature. The discussion of modelling formalisms and generalisability touches into the theoretical foundations of modelling and simulation. The impact of this thesis is directly related to the daily tasks of a model developer but extends outwards to organisational structure and decision making. The contribution of this thesis to Digital Twinning warrants reflection of this emerging technology's impacts on society, which are predicted to be widespread and varied.

7 Conclusion

The goal of this research was to develop an analytical approach to model decomposition. This resulted in the design of NewMODE, a network theory approach to model decomposition. The specification of the network-based metamodel in Section 4.1 provides an answer to RQ1; *"How can the architecture of a complex model be represented?"* The adapted GN algorithm in Section 4.2 addresses RQ2; *"How can the architectural metamodel be decomposed?"* This resulted in the design of ghost nodes to tackle inter-component synchronisation. Candidate solutions are evaluated on the size of their components, as well as the internal and external complexity. These criteria (Section 4.3) provide a means to evaluate the system as required by RQ3; *"How can model decompositions be evaluated quantitatively?"*

NewMODE was implemented for a modelling language, LSAT, from the manufacturing process modelling domain. Evaluation was performed by applying NewMODE to a monolith model of a high tech, complex manufacturing process. By comparing the results to a version of this model that had been manually decomposed by an expert, it was shown that NewMODE can achieve the same or better quality of decomposition and aids developers in re-architecting a monolith model.

7.1 Contribution

The main contribution of this thesis is the design of a completely novel approach to model decomposition. NewMODE fills an important research gap for an analytic approach to model decomposition that is automated and reproducible. Every aspect of this design is novel; the identification of the four decomposition steps, the network theory approach to metamodelling, and the use of the GN algorithm from social sciences. Given a legacy model in LSAT, a metamodel can be generated that represents the architecture of the monolith system. The GN algorithm for community detection in social networks can successfully decompose the monolith into a set of candidates, whereby each has highly cohesive and loosely coupled components. Using a directed calculation of the edge betweeness centrality respects the order of execution within the model, when decomposing it. Additionally, the recomposition of these components can also be performed by algorithmic identification of inter-component synchronisation requirements. It is possible to identify criteria that evaluate the quality of decomposition quantitatively. This depends heavily on the use case

and driving forces for decomposition in that scenario. The candidates identified provide a good base for further manual selection by a modeller and this selection can be guided by the decomposition criteria.

This study contributes to the discussion of the granularity trade-off in model decomposition as NewMODE produces candidates of various levels of granularity. This improves on the state-of-the-art as it offers users the opportunity to explore the granularity of the decomposition instead of relying on user input for the desired number of components. For the same level of granularity, the automated decomposition performs similar to the manual decompositions in size of components, but better than the manual decompositions in terms of link density and ghost density. This validates the results of the decomposition algorithm for a given level of granularity. Additionally, it is possible to explore the stopping condition for granularity using NewMODE, although the determination of a single candidate as the recommended decomposition architecture is dependent on aggregation of the multiple criteria.

Although model decomposition is motivated by the desire to manage the complexity of models, it equally results in insight to the physical process behind the model itself. The physical process is a complex system already and by building a model of it in a monolith-first approach, the decomposition reflects the subparts and structure of complexity in the physical machine which may be useful in its own right. The identification of ghost resources can be seen as an algorithmic solution to identifying implicit assumptions within the monolith model. This contributes to the challenge of dealing with the implicit assumptions within the model [26]. These inter-component synchronisations have semantic meaning in the real world, which was hidden in a synchronisation node in the model.

Formalisation of modelling techniques, such as model decomposition, contributes to the advancement of the discipline of modelling and simulation. In one sense it moves towards an automation of the processes which are tedious and time-consuming for modellers by providing them with tools that aid a better decomposition design. Aside from automation, formalising modelling techniques also allows the manual process to be clearly defined. This has important practical implications for ease of knowledge transfer in the modelling domain and importantly facilitates reflection on manual decomposition to enable improvement. The demand for model management is growing in industry, accelerated by the desire to digitalize their systems and adopt emerging technology like Digital Twin. Formal modelling techniques can contribute to model management in academics and industry.

More generally, this thesis contributes to the management of complexity; an increasing challenge as we become better and faster at building software, models, generating data and online systems. The need to be able to manage these systems at the same rate that we can generate them is a challenge to which there is no easy answer. Although modularity is a concept that can theoretically sound like a solution to complexity in whatever form it presents itself (object oriented design, components based modelling, microservices etc.) it always involves separating the modules from the interface that connects them. Model decomposition is a task of analysing complexity itself in such a way that allows us to hide the complexity as much as possible, or control the components in which it resides. I consider this analytical approach to model decomposition a contribution to the long standing challenge of system modularity.

7.2 Future work

This thesis describes a network-based approach for model decomposition using metamodelling and algorithmic detection of components. Although this has been applied to the specific formalism of LSAT, the following analyses would be valuable next steps. Firstly, running this same implementation on many more LSAT models to get a more general result for the usefulness of the methodology. This also is expected to provide further insight into useful criteria for decomposition that distinguish between different model topologies. Similarly, this implementation could be adapted to other model formalisms in the class of logistics and process modelling. This would involve defining a transformation from the source formalism to the metamodel described in this methodology.

The decomposition process proposed, relies on the Girvan-Newman algorithm for community detection in networks. It would be useful to compare this process while replacing the decomposition algorithm with other hierarchical clustering algorithms. Of course, this would require adapting them to be suitable for a directed, acyclic graph. If a stochastic algorithm were implemented, the iteration between criteria and decomposition would essentially become a multiple-objective optimization problem which might yield interesting results. Another suggestion to improve the methodology is to implement hierarchical decomposition on components are particularly large in a candidate that is otherwise performing well in all the criteria.

There are also recommendations for advancing research into model decomposition more generally. Developing a topology of standard structures that can be used as a testbed for future research on model decomposition. This might consist of 'toy models' that are available in pairs of monolith and modular architecture. This would involve future work to explore the characteristics of decomposed models and prove that certain structures are sufficient as testbeds.

Bibliography

References

[1] Carliss Y. Baldwin and Kim B. Clark. *Design Rules: The Power of Modularity Volume 1*. 1999.

[2] Armin Schwienbacher Benjamin Larralde. Electronic copy available at : http://ssrn.com/abstract=2545489 Electronic copy available at :. *Grou*, 23529(2):1–45, 2008.

[3] Chris Richardson. *Microservices Patterns*. Manning Publications, NY, 1st edition, 2019.

[4] Ralf Mosshammer, Friederich Kupzog, Mario Faschang, and Matthias Stifter. Loose coupling architecture for co-simulation of heterogeneous components. *IECON Proceedings (Industrial Electronics Conference)*, pages 7570–7575, 2013.

[5] Deniz Cetinkaya and Alexander Verbraeck. Metamodeling and model transformations in modeling and simulation. *Proceedings - Winter Simulation Conference*, (December):3043–3053, 2011.

[6] Sara Hassan and Rami Bahsoon. Microservices and their design trade-offs: A self-adaptive roadmap. *Proceedings - 2016 IEEE International Conference on Services Computing, SCC 2016*, pages 813–818, 2016.

[7] Davide Taibi and Kari Systä. A Decomposition and Metric-Based Evaluation Framework for Microservices. 2019.

[8] Michael Grieves. Origins of the Digital Twin Concept. *Revista de obstetricia y ginecolog??a de Venezuela*, 23(August):889–896, 2016.

[9] Adil Rasheed, Omer San, and Trond Kvamsdal. Digital Twin: Values, Challenges and Enablers. pages 1–31, 2019.

[10] Roland Rosen, Stefan Boschert, and Annelie Sohr. Next Generation Digital Twin. *Atp Magazin*, 60(10):86, 2018.

[11] Murat Gunal. *Simulation for Industry 4.0. Past, Present, and Future*. Springer Nature Switzerland, 1st edition, 2019.

[12] Edmund Widl, Wolfgang Muller, Atiyah Elsheikh, Matthias Hortenhuber, and Peter Palensky. The FMI++ library: A high-level utility package for FMI for model exchange. *2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems, MSCPES 2013*, 2013.

[13] Stefan Boschert and Roland Rosen. Digital Twin - The Simulation Aspect. In P Hehenberger and D Bradley, editors, *Mechatronic Futures*, pages 59–74. Springer, Cham, 2016.

[14] Herbert A. Simon. The Architecture of Complexity. *The Roots of Logistics*, 106(6):467–482, 1962.

[15] Yilin Huang, Mamadou D Seck, and Alexander Verbraeck. From data to simulation models: component-based model generatin with a data-driven approach. *Proceedings of the 2011 Winter Simulation Conference*, (2010):3724–3734, 2011.

[16] Christina Terese Joseph and K. Chandrasekaran. Straddling the crevasse: A review of microservice software architecture foundations and recent advancements. *Software - Practice and Experience*, 49(10):1448–1484, 2019.

[17] Muhammad Abdullah, Waheed Iqbal, and Abdelkarim Erradi. Unsupervised learning approach for web application auto-decomposition into microservices. *Journal of Systems and Software*, 151:243–257, 2019.

[18] Charles G. Sanders. Research into cloud-based simulation: A literature review. *2019 Simulation Innovation Workshop, SIW 2019*, (February), 2019.

[19] Sam Newman. *Building Microservices.* O'REilly Media, Inc., 1st edition, 2015.

[20] Hans Vangheluwe and Juan De Lara. Meta-models are models too. *Winter Simulation Conference Proceedings*, 1:597–605, 2002.

[21] Sara Hassan, Nour Ali, and Rami Bahsoon. Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity. *Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017*, (April 2018):1–10, 2017.

[22] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

[23] Gabor Kecskemeti, Attila Csaba Marosi, and Attila Kertesz. The ENTICE approach to decompose monolithic services into microservices. *2016 International Conference on High Performance Computing and Simulation, HPCS 2016*, pages 591–596, 2016.

[24] Robert E. Shannon. Introduction to the art and science of simulation. *Winter Simulation Conference Proceedings*, 1:7–14, 1998.

[25] Hugo Brunelière, Jordi Cabot, Grégoire Dupé, and Frédéric Madiot. MoDisco: A model driven reverse engineering framework. *Information and Software Technology*, 56(8):1012–1032, 2014.

[26] Marko A. Hofmann. Criteria for Decomposing Systems Into Components in Modeling and Simulation: Lessons Learned with Military Simulations. *Simulation*, 80(7-8):357–365, 2004.

[27] A. Hochrein. From Monolith to Micro-Services. In *Designing Microservices with Django*, chapter 5, pages 111–137. 2019.

[28] Shanshan Li, He Zhang, Z. Jia, Z. Li, C. Zhang, J. Li, Q. Gao, Jidong Ge, and Zhihao Shan. A dataflow-driven approach to identifying microservices from monolithic applications. *Journal of Systems and Software*, 2019.

[29] R Gauthier and P Stephen. *Designing systems progams (automatic computing*. Prentice Hall, 1970.

[30] R.C. Martin, J.M. Rabaey, A.P. Chandrakasan, and B Nikolic. SRP: The Single Responsibility Principle. In *Agile Software Development*, chapter 8. Pearson Education, 2 edition, 2003.

[31] W.P. Stevens, G.J. Meyers, and L.L. Constantine. Structured Design. *IBM Systems Journar*, 13:115–139, 1974.

[32] Michael Gysel, Lukas Kölbener, Wolfgang Giersche, and Olaf Zimmermann. Service cutter: A systematic approach to service decomposition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9846 LNCS:185–200, 2016.

[33] Genc Mazlami. Algorithmic Extraction of Microservices from Monolithic Code Bases. page 113, 2017.

[34] S Rugaber and K Stirewalt. Model-driven reverse engineering. *IEEE Software*, 21:45–53, 2004.

[35] Yves R Schneider and Anne Koziolek. Towards Reverse Engineering for Component-Based Systems with Domain Knowledge of the Technologies Used. 2019.

[36] Daniel Escobar, Diana Cardenas, Rolando Amarillo, Eddie Castro, Kelly Garces, Carlos Parra, and Rubby Casallas. Towards the understanding and evolution of monolithic applications as microservices. *Proceedings of the 2016 42nd Latin American Computing Conference, CLEI 2016*, 2017.

[37] Ervin Djogic, Samir Ribic, and Dzenana Donko. Monolithic to microservices redesign of event driven integration platform. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, (May):1411–1414, 2018.

[38] Pooyan Jamshidi, Claus Pahl, and Nabor C. Mendonça. Pattern-based multi-cloud architecture migration. *Software - Practice and Experience*, 47(9):1159–1184, 2017.

[39] L (Politecnico di Milano) Baresi, A (National University of Comahue) Derenzis, and M (Politecnico di Milano) Garriga. Microservices Identification Through Interface Analysis. In Flavio (University of Milano-Bicocca) de Paoli, Stefan (Vienna University of Technology) Schulte, and Einar (University of Oslo) Broch Johnsen, editors, *Service-Oriented and Cloud Computing. ESOCC 2017. Lecture Notes in Computer Science*, volume 10465, chapter Microsevic, pages 19–33. Springer Nature, Cham, 2017.

[40] Alessandra Levcovitz, Ricardo Terra, and Marco Tulio Valente. Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems. 2016.

[41] Tanwir Ahmad, Junaid Iqbal, Adnan Ashraf, Dragos Truscan, and Ivan Porres. Model-based testing using UML activity diagrams: A systematic mapping study. *Computer Science Review*, 33:98–112, 2019.

[42] Ferdinand Wagner. *Modeling Software with Finite State Machines*. 2006.

[43] Ronald Calinger. From the Problem of the Seven Bridges of Konigsberg. In *Classics of Mathematics*. Englewood Cliffs, N.J. : Prentice Hall, ©1995, 1995.

[44] Wai-Kai Chen. Foundations of Network Theory. *Broadband Matching*, pages 1–47, 2015.

[45] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Community structure in social and biological networks. *PNAS*, 99(12):7821–7826, 3 2004.

[46] Leo Speidel, Taro Takaguchi, and Naoki Masuda. Community detection in directed acyclic graphs. *European Physical Journal B*, 88(8), 2015.

[47] M. Girvan and M. E.J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):7821–7826, 2002.

[48] ESI. L-SAT.

[49] Bram Van Der Sanden, Michel Reniers, Marc Geilen, Twan Basten, Johan Jacobs, Jeroen Voeten, and Ramon Schiffelers. Modular model-based supervisory controller design for wafer logistics in lithography machines. *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems, MODELS 2015 - Proceedings*, pages 416–425, 2015.

[50] Wim H. Hesselink and Mark Ijbema. Starvation-free mutual exclusion with semaphores. *Formal Aspects of Computing*, 25(6):947–969, 2013.

[51] Olivier L. de Weck. Multiobjective Optimization : History and Promise. *in Proc. 3rd China-Japan-Korea Joint Symp. Optimization Structural Mech. Syst. Invited Keynote Paper GL2-2*, (January 2004):14, 2004.

[52] Caner Hamarat, Jan H. Kwakkel, Erik Pruyt, and Erwin T. Loonen. An exploratory approach for adaptive policymaking by using multi-objective robust optimization. *Simulation Modelling Practice and Theory*, 46:25–39, 2014.

[53] John M. Jeffrey. Using Petri nets to introduce operating system concepts. *ACM SIGCSE Bulletin*, 23(1):324–329, 1991.

[54] Wu Qinyi, Calton Pu, and Akhil Sahai. DAG synchronization constraint language for business processes. *CEC/EEE 2006 Joint Conferences*, 2006(Section 3):4–11, 2006.

[55] Shannon Flumerfelt. *Franz-Josef Kahlen, Shannon Flumerfelt, Anabela Alves (eds.)-Transdisciplinary Perspectives on Complex Systems_ New Findings and Approaches-Springer International Publishing (2017).pdf*. 2017.

[56] Hendrik van der Valk, Hendrik Haße, Frederik Möller, Michael Arbter, Jan-Luca Henning, and Boris Otto. A Taxonomy of Digital Twins. *26th Americas Conference on Information Systems (AMCIS)*, (August):1–10, 2020.

[57] Richard M. Fujimoto, Nurcin Celik, Haluk Damgacioglu, Michael Hunter, Dong Jin, Young Jun Son, and Jie Xu. Dynamic data driven application systems for smart cities and urban infrastructures. *Proceedings - Winter Simulation Conference*, 0(Oecd 2011):1143–1157, 2016.

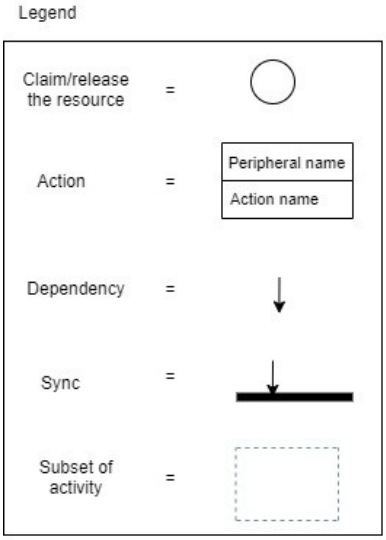
[58] Mengmeng Yang, Jijia Li, Yiran Man, Zijun Peng, Xiaofang Zhang, and Xudong Luo. Integration of an Energy Management Tool and Digital Twin for Coordination and Control of Multi-vector Smart Energy Systems. *Materials & Design*, page 108334, 2019.

[59] Peter Augustine. The industry use cases for the Digital Twin idea. *Advances in Computers*, 117(1):79–105, 2020.

A

LSAT example

A.1 Activity flow diagram



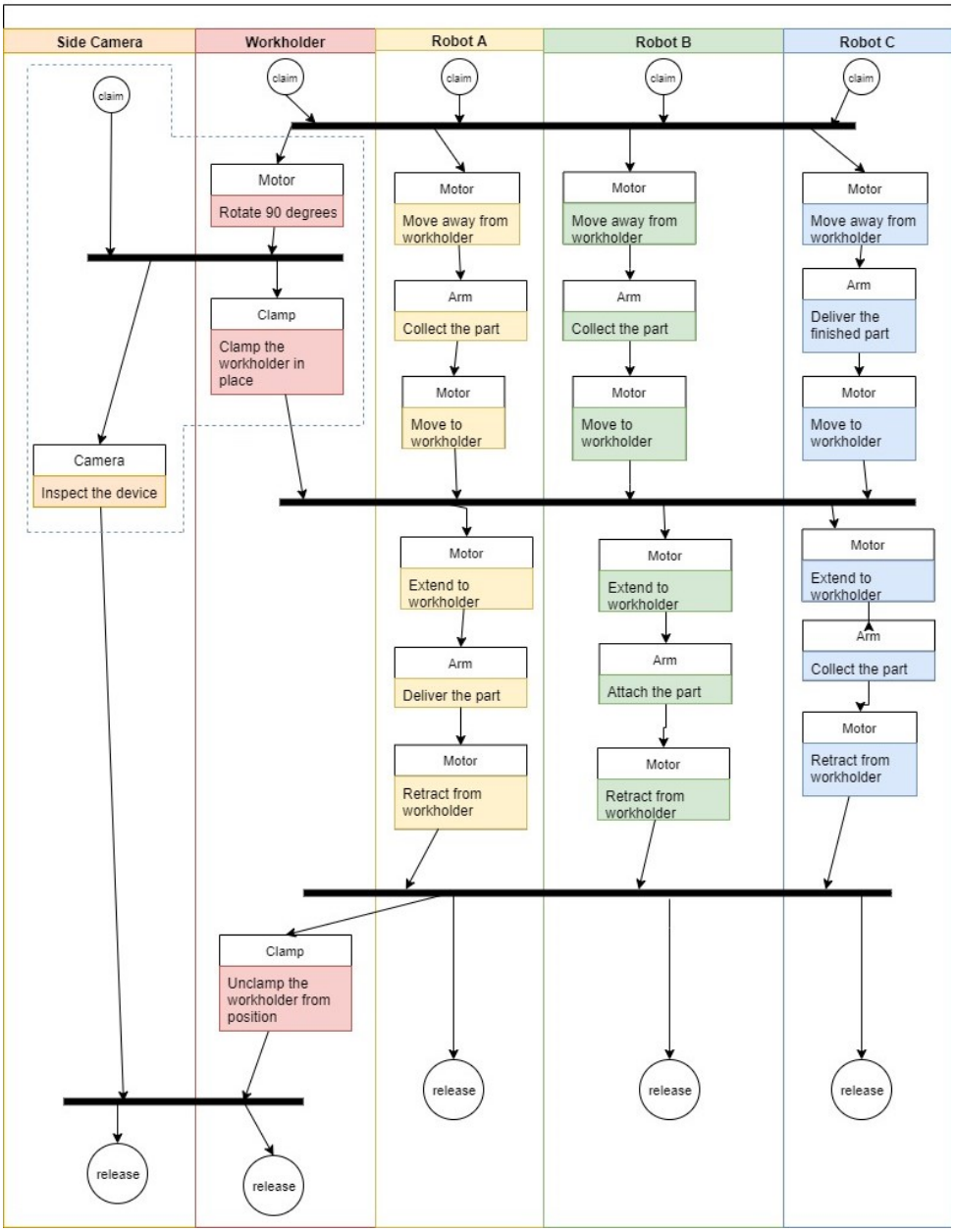


Figure A.1: Complete activity flow for running example.

B

Metamodel example

B.1 Example metamodel: layer 1

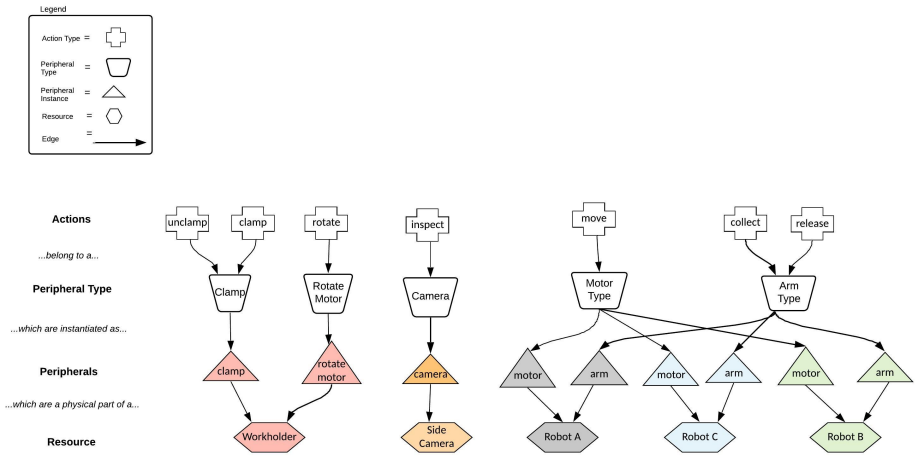


Figure B.1: Fully specified layer 1 of the metamodel for the example machine

C

Implementation of NewMODE

C.1 NewMODE dependencies

NewMODE depends on the following:

- Python version 3.7.3 +
- The following python packages: networkx, numpy, pandas, statistics, seaborn, scipy, matplotlib, itertools, random, re, os, shutil, dateutil, collections

C.2 Example code

```
from metamodel import Metamodel
import LSATconnect
import decomposition
import analysis


if __name__ == '__main__' :

    #initialise the metamodel object
    machine = Metamodel("Monolith example")

    #load in the data from the LSAT files
    machine.specs  = LSATconnect.extractModel(srcDir)

    #Create the metamodel layers
    machine.createLayer1()
    machine.createLayer2()
    machine.createLayer3()

    #Run the decomposition algorithm.
    #Candidates is a dict with Key = id and value = list of action instances in that component
    candidates = decomposition.runDecomposition(machine, lowerLimit, upperLimit)

    #evaluate the candidates
    componentCriteria, systemCriteria = analysis.evaluateCriteria(candidates)

    #return the IDs of candidates on the Pareto front
    paretoIDs = analysis.getPareto(candidates)

    #visualise criteria
    analysis.plotCriteria(componentCriteria, systemCriteria)

    #To plot a network metamodel within Python
    nx.draw_networkx(machine.graph, with_labels = True)

```

Figure C.1: Example of the code needed to use NewMODE.

D

Results

D.1 Size of components

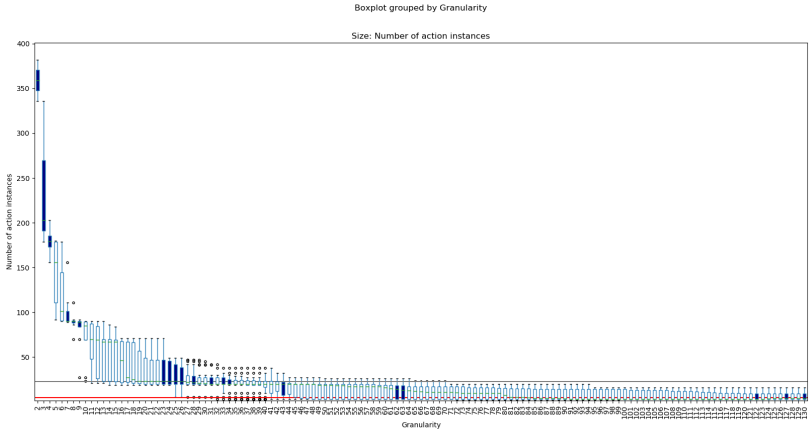


Figure D.1: size vs granularity boxplot on a linear scale for all levels of granularity.

D.2 Mean link density

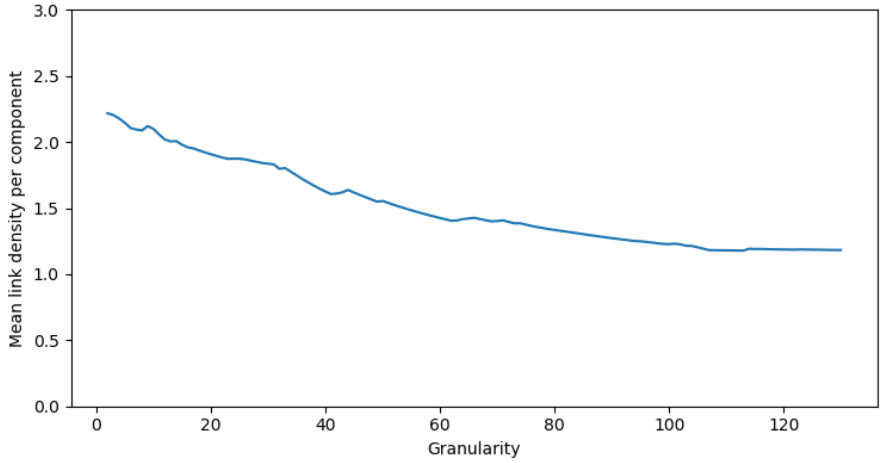


Figure D.2: Mean link density (linear scale) plotted against the level of granularity.

D.3 Distribution of link density

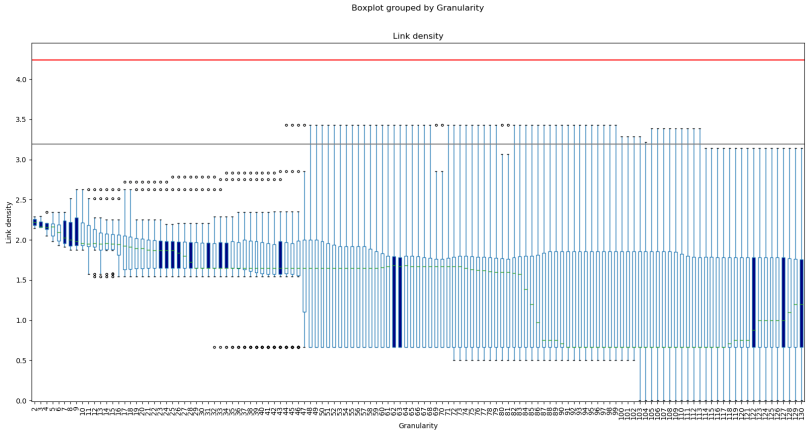


Figure D.3: Comparing the distribution of internal links in each component. The x-axis is the granularity and every candidate is plotted as one boxplot. For a single candidate, you can see the distribution of the size of each of its components in the boxplot. Alternatively, it is possible to see the mean of the size vs granularity by following the green line that runs through each boxplot. Additionally, there are two reference lines included in this plot. The grey horizontal line represents the value of this criteria for the monolith model A. The red horizontal line represents the value of this criteria for the modular model B. Pareto optimal candidates are navy.

D.4 Mean ghost density

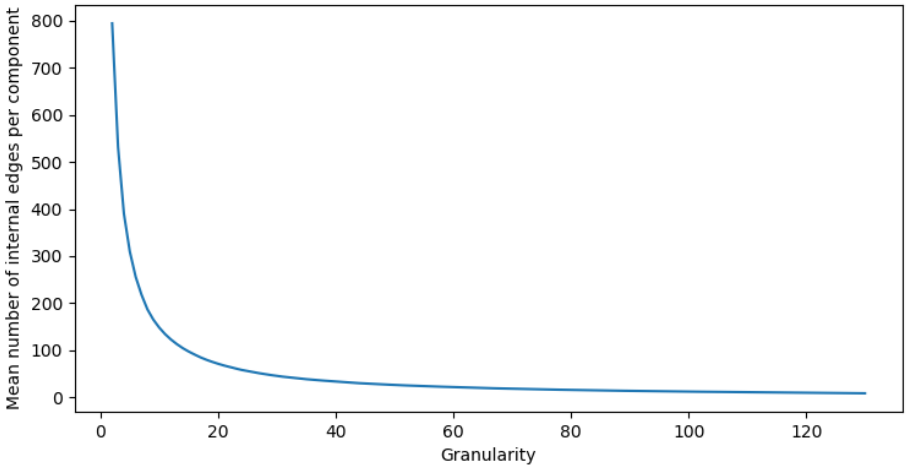


Figure D.4: Mean ghost density (linear scale) plotted against the level of granularity.

D.5 Distribution of ghost density

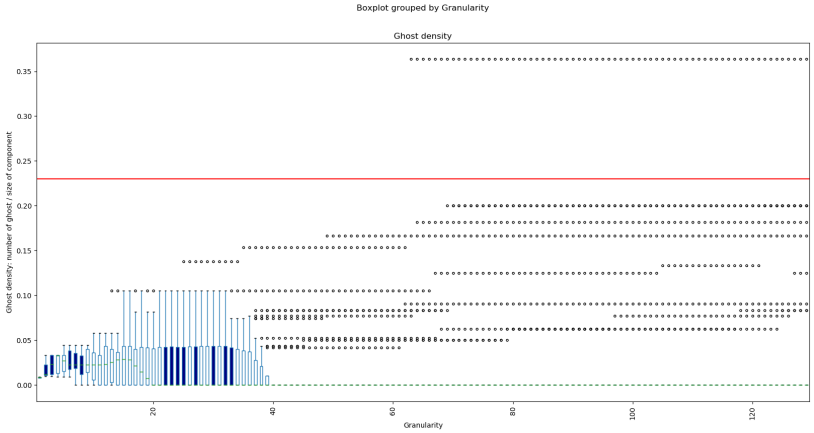


Figure D.5: Comparing the ghost density in each component. The x-axis is the granularity and every candidate is plotted as one boxplot. For a single candidate, you can see the distribution of the size of each of its components in the boxplot. Alternatively, it is possible to see the mean of the size vs granularity by following the green line that runs through each boxplot. Additionally, there are two reference lines included in this plot. The grey horizontal line represents the value of this criteria for the monolith model A. The red horizontal line represents the value of this criteria for the modular model B. Pareto optimal candidates are navy.

D.6 Number of ghosts

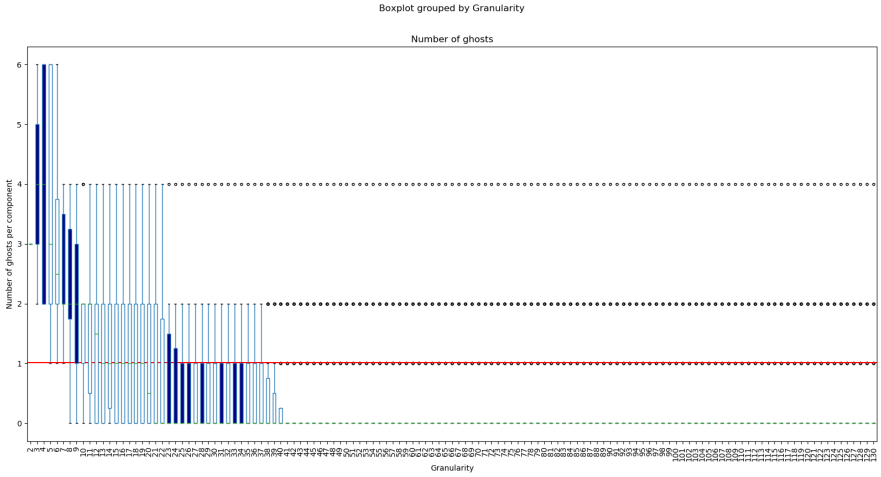


Figure D.6: Comparing the number of ghosts in each component, for all candidates. The x-axis is the granularity and every candidate is plotted as one boxplot. For a single candidate, you can see the distribution of the size of each of its components in the boxplot. Alternatively, it is possible to see the mean of the size vs granularity by following the green line that runs through each boxplot. Additionally, there are two reference lines included in this plot. The grey horizontal line represents the value of this criteria for the monolith model A. The red horizontal line represents the value of this criteria for the modular model B. Pareto optimal candidates are navy.