Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

GM2MS4: A Transformation Tool from Goal-Oriented Models to System-of-Systems Mission Simulation

Manoel Vieira Coelho Neto

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientadora
Prof.ª Dr.ª Genaina Nunes Rodrigues

Brasília
2021

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Engenharia da Computação

Coordenador: Prof. Dr. Joao José Costa Gondim

Banca examinadora composta por:

Prof.ª Dr.ª Genaina Nunes Rodrigues (Orientadora) — CIC/UnB
Prof. Dr. Edison Ishikawa — CIC/UnB
Prof. Dr. Marcelo Antônio Marotta — CIC/UnB

CIP — Catalogação Internacional na Publicação

Coelho Neto, Manoel Vieira.

GM2MS4: A Transformation Tool from Goal-Oriented Models to System-of-Systems Mission Simulation / Manoel Vieira Coelho Neto. Brasília : UnB, 2021.
91 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2021.

1. System-of-Sytems, 2. modelagem de SoS, 3. simulação de SoS, 4. conversão de modelos, 5. modelagem orientada a objetivos e missões de SoS

CDU CC672g

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

GM2MS4: A Transformation Tool from Goal-Oriented Models to System-of-Systems Mission Simulation

Manoel Vieira Coelho Neto

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof.ª Dr.ª Genaina Nunes Rodrigues (Orientadora)
CIC/UnB

Prof. Dr. Edison Ishikawa
CIC/UnB

Prof. Dr. Marcelo Antônio Marotta
CIC/UnB

Prof. Dr. Joao José Costa Gondim
Coordenador do Curso de Engenharia da Computação

Brasília, 16 de Novembro de 2021

Por um mundo onde sejamos socialmente iguais, humanamente diferentes e totalmente livres

Rosa Luxemburgo

Agradecimentos

Agradeço à minha família pelo esforço hercúleo empenhado em minha educação com o objetivo de que estas linhas um dia fossem escritas. Agradeço ao suporte que me foi dado nos dias mais difíceis, principalmente pelos meus pais e tios, necessários para minha manutenção no curso.

Agradeço aos meus pais por terem ensinado que o melhor caminho para avançar é o estudo e a priorizá-lo acima de todas as coisas. Agradeço pelas inúmeras vezes que me incentivaram a ir até o final e a não desistir no meio do caminho. Agradeço às minhas irmãs, Anna e Maria, por estarem sempre presentes e pelo seu afeto.

Agradeço ao irmão e tia que o universo me deu, Davi e Sandra Ory, por estarem sempre presentes nas dificuldades e por sempre me ajudarem a superá-las.

Sair de casa e deixar o conforto da família em busca de novas oportunidades - apesar de estimulante num primeiro momento aos olhos de um garoto - pode ser um desafio muito maior que o esperado. Por isso, agradeço àqueles que, mesmo sem ligação sanguínea, me tomaram como parte dos seus e me acolheram, prestando uma contribuição enorme para que eu chegasse aqui. Sem essas pessoas jamais conseguiria ter superado os obstáculos encontrados nos últimos anos. Agradeço cada carona, cada RU, cada almoço de domingo. Sem o acolhimento e suporte de vocês este trabalho não poderia ter sido finalizado.

Agradeço à equipe UnBall por todo conhecimento que construímos juntos e aprendizados que tivemos como time. Meus anos na equipe foram essenciais para o meu progresso acadêmico e profissional. Agradeço pelos laços que fizemos ali e que carrego sempre comigo.

Agradeço aos meus amigos Izabella e Martin por terem me ajudado na revisão deste trabalho.

Por fim, agradeço à minha orientadora Prof.ª Dr.ª Genaina Nunes Rodrigues e à minha tutora Prof.ª Dr.ª Vanessa Tavares Nunes, pela oportunidade oferecida e pelo suporte durante o caminho que percorremos. Por terem me estimulado durante o desenvolvimento e escrita e por terem acreditado no sucesso deste trabalho.

Resumo

System-of-Systems (SoS, ou em português: Sistema-de-Sistemas) são aqueles sistemas resultantes da integração de outros sistemas independentes (sistemas constituintes). A modelagem desses sistemas através de técnicas tradicionais de design de sistemas é uma tarefa árdua, levando em consideração as diversas formas que estes sistemas contituintes podem participar em processos de SoS. Há algumas propostas de modelo na literatura, como o projeto mKaos[14], mas esses modelos não podem ser simulados ou conectados a aplicações externas à simulação para que seja validado. A partir desse ponto, o presente trabalho propõe um modelo orientado a objetivos para SoS que: (i) foque nas missões que um SoS deve cumprir ao invés da visão arquitetural do sistema e (ii) possa ser convertido em um projeto MS4 capaz de simular os comportamentos extraídos do modelo de objetivos. Adicionalmente o designer poderá conectar o modelo MS4 a aplicações existentes para a execução da missão possa ser verificada. Uma ferramenta para essa conversão de modelos (GM2MS4) é provida juntamente a este manuscrito e é capaz de transformar o modelo de missões proposto em uma simulação MS4 que verifique o cumprimento da missão.

Palavras-chave: System-of-Sytems, modelagem de SoS, simulação de SoS, conversão de modelos, modelagem orientada a objetivos e missões de SoS

Abstract

System-of-Systems (SoS) are those systems resulted by the integration of other independent systems (constituent systems). Its modeling through traditional system design approaches is a challenging tasks, considering the multiple ways that these constituent systems may participate on the SoS processes. There are some SoS model proposals on the literature, such as the mKaos[14] project, but these models cannot be simulated neither can be connected to external applications in order to validate it. Starting from this point, this work proposes a goal-oriented model for SoS that: (i) focuses on the missions that an SoS may accomplish, rather than its architectural view and (ii) can be converted into an MS4 project capable of simulating the behaviors extracted from the goal-model. Additionally, the designer may connect the MS4 model to existing applications in order to verify the mission execution. A tool for this model conversion (GM2MS4) is provided along with this manuscript and is capable of transforming the mission model proposed into an MS4 simulation that verifies the mission accomplishment.

Keywords: System-of-Systems, SoS, SoS modeling, MS4, SoS simulation, model conversion, goal-oriented modeling and SoS missions

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

With the increase of autonomous interconnected systems over the last years and the integration of cyberphysical systems into daily life, a new kind of complex system category emerged: the System-of-Systems [8]. It refers to those systems made out of other independent systems, where the whole is not just the sum of the parts as in the design of common systems. Rather, these *constituent systems* have their goals and autonomy to choose if they will contribute to the System-of-Systems (also known as SoS) purposes or chase their own goals instead. Also, new features may emerge by connecting those constituent systems together to accomplish some major goals.

While systems which needed to be integrated to other systems are very popular today (microservices architecture, IoT, and blockchain solutions are a few examples), the architects of these systems tend to still model them in a traditional way, and they often may disregard the emergent behaviors that these independent systems could provide while working in a group setting. This lack of focus on emergent behavior, or rather the attempt to restrict them, happens because the traditional approach of systems design focuses mainly on the architectural view of the system, and therefore it is difficult to fit the emergent behaviors into this modeling perspective.

We search then, for a model that focuses on the missions (or the goals) of a System-of-Systems, one that can not only abstract the architectural view of the system but also allow the design to validate the model by connecting its simulation to existing systems. In other words, we focus on a high-level perspective of the system capable of evaluate the behaviors of the SoS and the behaviors that emerge from a global mission (or a set of goals) accomplishment. By "global mission" we mean the connection setting of the constituent systems. If a simulation of the model is provided, it may evaluate the mission execution, outputting if the modeled mission is achievable or not.

1.2 Problem and Hypothesis

The problem aimed is the lack of a model that describes a System-of-Systems from a high-level perspective (missions) and its behaviors, rather than focusing only on its structural view. Based on the research shown in the Chapter 2, it was stated that:

1. There are mission model specifications existent in the literature (such as the mKaos [14]). Yet, those models are concerned with the expected behaviors and their description, serving as a design document. Therefore, we want to assess if it is possible to validate a model that extends such specification by allowing the designer to implement the tasks present at the model.
2. There are software that allows System-of-Systems validations from a behavioral view, such as the MS4 ME presented later in the section 2.3.1.

Unfortunately, the simulator won't simulate pure mission (or goal) models, neither the models found can be simulated in any other tool. The question that arises here is: Is it possible to have a tool capable of converting from a given mission model to the MS4 model (enabling then the mission to be validated through its simulation)? Such tool is provided in this work.

The hypothesis that was tested in this work is that a goal model provide this high-level abstraction, keeping it as close as possible to the model proposed in mKaos [14], this model should be convertible into an MS4 project and validated through the simulation of this generated project. It is expected that, from the mission model proposed an equivalent MS4 project could be able to provide enough information for the design to evaluate whether a mission is achievable and to point out emergent behaviors on that System-of-Systems that allow the mission to be accomplished, or even behaviors that prejudices the mission accomplishment.

1.3 Goals

1.3.1 General goals

Our general goals are:

- To provide a model that specifies a mission for a System-of-Systems; and
- To develop a tool capable of converting this mission model into an MS4 project allowing the designer to implement the tasks presented at the model by interfacing the simulation with the real world.

1.3.2 Specific goals

To achieve this model specification and integration we focus on four main objectives as described below:

1. To provide a goal model that has the necessary elements to specify a SoS mission and the constituent systems relations present at this mission accomplishment.
2. To provide a software solution that is capable of transforming this model into an MS4 simulation, expressing the expected behaviors presented at the mission model in the face of each constituent system and how they are interconnected.
3. To verify those behaviors by adding the tasks presented at the mission model as implementable code on the MS4 resulting project, allowing the designer to interface the simulation with existent systems or self-implemented mocks, and then to validate the mission accomplishment.
4. To illustrate the model proposal by modeling a mission for a private blockchain network.
5. To test the proposed model and toolchain by implementing behaviors on the code of the tasks and verifying if the simulation result corresponds to the expected execution expressed in the model.

1.4 Organization

This manuscript is organized into another four chapters as described next.

In Chapter 2 it is presented the baseline of the research for this work, showing the works that served as basis for the proposal. Some major points are covered:

- System-of-Systems definition and what are its differences to common systems;
- Properties from the goal-oriented approach that can be associated with System-of-Systems characteristics;
- Existent works that guided the model proposal;
- System-of-Systems simulation and what is the necessary data to construct it;
- Introduction to the domain used as an example for the model construction. Explaining which characteristics were assumed for the modeled blockchain network, as for its constituent systems and their properties.

Chapter 3 shows how the proposal was developed, providing the methodology of this work and how the results could be achieved. This section gives detailed information for the reader to comprehend the process executed to convert the model into a simulation and verify the results. Also, the instructions for the model creation and conversion as for its simulation execution are provided here.

In Chapter 6, the simulation results and the verification for the example model are shown. We analyzed and evaluated the following questions: What is the execution result for an implemented model?; Does the simulation fail when behaviors are not attended by the implementation of the tasks?; and What happens when an OR-linked task fails?.

Lastly, the conclusions are presented in Chapter 7.

Chapter 2

Research Baseline

In this chapter, we show the knowledge base and methods used as a basis to develop our proposal. It is divided as follows: First, in Section 2.1 we introduce the system-of-systems concept regarding its definitions and its differences from common systems. In Section 2.2, we show why a System-of-Systems can be modeled under a goal-oriented approach and the modeling tool we have chosen for this project. In Section 2.3, we present the simulation tool that may validate the proposed model. In the last section 2.4, it is explained why a blockchain is an SoS and thus can be a valid domain to exemplify the proposal.

2.1 System-of-systems

A system is usually seen as a whole, as a black-box that must meet some criteria, and by doing so, it is defined as a *functional* system. In other words: "A system is a set of interrelated elements" [1]. But very often those elements are other systems by themselves, which are then called *constituent systems*. The resulting set created by grouping other individual and autonomous systems is defined as a "system-of-Systems" and it may appear abbreviated as **SoS** from now on in this manuscript.

The question that arises is: What distinguishes a system composed of *systems* from a system composed of *parts*? The following subsections will answer that question by looking at the SoS definition and evidencing some distinctions between them.

2.1.1 Definition

The most commonly used definition for an SoS is the one presented by Maier [8] who states that "a System-of-Systems is an assemblage of components which individually may be regarded as systems, and which possesses two additional properties: operational independence of components and managerial independence of components".

For the first additional property, the subsystems must guarantee that once they are disassembled from the mesh they must continue to operate independently (although the disconnected element probably would be pursuing its own goals, rather

than the mesh's one). As for the second, "the components not only *can* operate independently, they *do* operate independently"[8].

2.1.2 Distinguishing a System from a System-of-Systems

For a system to be characterized as a System-of-Systems the literature converges[3] to some properties that must be shown to exist in the system design.

We can distinguish a *system*, properly speaking, from an SoS by comparing them using the following characteristics[3]:

Autonomy

- System: parts often lose autonomy to grant autonomy to the system
- System-of-Systems: constituent systems must be **autonomous** to accomplish the purpose of SoS

Belonging

- System: parts are intrinsic to the system itself and designed as such, the parts are not independent.
- System-of-Systems: constituent systems are **independent** and because of that, they may or may not choose to **contribute** to accomplish the SoS purposes, based on its own goals and taking into account the SoS' goals on that decision process.

Connectivity

- System: all the connections are known at the system's design time, multiple connections are abstracted inside the elements, and the external connections to other systems are minimized.
- System-of-Systems: All the constituent systems **may communicate to each other** allowing them to contribute to each other to accomplish their individual goals.

Diversity

- System: Simpler abstractions are implemented to reduce the diversity of the parts, forcing different discrete modules to fit in known interfaces.
- System-of-Systems: The open and dynamic connectivity allows the constituent systems to be more diverse and even **play different roles for different contexts** in an SoS.

Emergent Behaviours

- System: for a simple system, its emergence is foreseeable (or at least there is an attempt to predict it by brainstorming, testing, etc.) and must be designed or tested to avoid any kind of unexpected emergent behaviors.
- System-of-Systems: the emergent behaviors of the constituent systems are **deliberately not designed**, neither can be predicted. Instead, an additional capability in the SoS, provided by some of the constituent systems, would detect and warn/eliminate bad behaviors from the mesh.

2.2 Goal-Oriented SoS Modeling

In this Section, we present the concepts for a goal-seeking system and how it relates to the SoS modeling. In other words: What is the basis to model an SoS from a goal-oriented modeling approach? The following subsections show that an SoS can be modeled as a goal-seeking system, and thus it is possible to model it through its goals rather than its architecture.

2.2.1 Goal-seeking system

Ackoff [1, p.665] states that:

> "A *goal-seeking* system is one that can respond differently to one or more different external or internal events in one or more different external or internal states and that can respond differently to a particular event in an unchanging environment until it produces a particular state (outcome). Production of this state is its goal. Thus **such a system has a choice of behavior**. **A goal-seeking system's behavior is responsive, but not reactive**. A state which is sufficient and thus deterministically causes a reaction cannot cause different reactions in the same environment.
>
> Under constant conditions **a goal-seeking system may be able to accomplish the same thing in different ways and it may be able to do so under different conditions**."Ackoff [1, p.665]

It is easy to conclude that an SoS satisfies the properties required by this definition of a *goal-seeking system*, and therefore it can be modeled as one. A goal-oriented approach would allow us to express the variability of an SoS through the relations between the goals and their dependencies. We can then verify if there is any variability. If the supposed SoS have different paths to achieve the same goal, it may be an SoS, otherwise, an SoS approach is not recommended and more traditional modeling methods may be used. Also, based on the requirements of the goals we can check if a capability is available on the network.

Goal-oriented modeling expresses the possible behaviors for goal achievement. A declarative modeling approach like this makes the expected behaviors and possibilities explicit, instead of trying to guess them from a port-interface diagram. By going this way we believe we can raise the abstraction level of the set, from a

pure relational perspective to a behavioral one. We believe that this can reduce the (already high) complexity of the requirement gathering for an SoS and its mission execution analysis.

2.2.2 The mKaos project

This kind of perspective over SoS design is not entirely new and the mKaos [14] first proposed a modeling language for SoS based on the Kaos[7] abstraction for *goals*, but extending it to *missions* of SoS, its conceptual model is shown at the Figure 2.1.

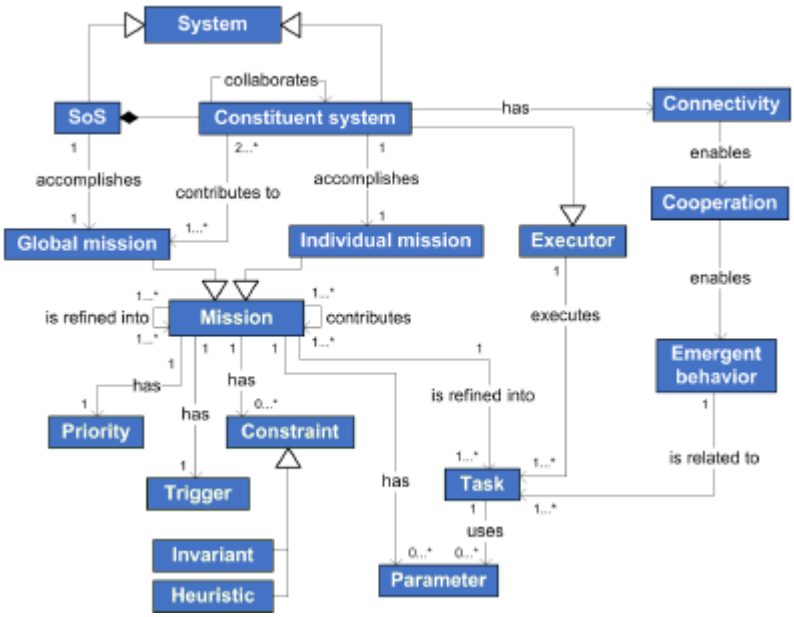

Figure 2.1: mKaos conceptual model for missions of SoS [15]

That work suggests some elements that constitute a mission in an SoS context. Those are:

- Tasks

  A task is an operation to be executed by the system, these are the final decomposition for a goal, a task must represent something that happens at one of the systems, the decomposition for a task is organized to follow a specific order. As shown in the Figure 2.2

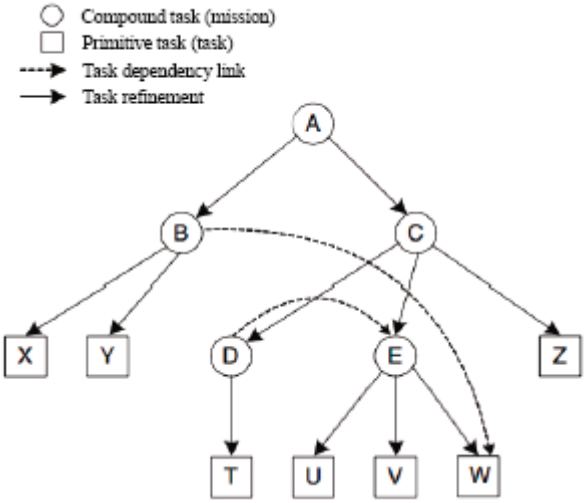

Figure 2.2: A tree representation of a task composition [14]

- Preconditions

  Also called *trigger*, this is a set of conditions (operational limitations or requirements) that must make the system to execute a specific mission.

- Executor

  The executor is a system of the SoS that executes a *task*.

- Priority

  The priority that a mission has over another in the SoS.

- Parameter

  Parameters are variables that a mission can receive or return, they can offer additional knowledge about the context or provide conditions for a mission or tasks to execute.

- Constraints

  Additional conditions must be satisfied over the system execution, if not satisfied the mission may fail. The constraints can be classified as *invariants* or *heuristics*. The first refers to the mandatory conditions that an SoS must meet, as the second, refers to the conditions whose satisfaction is not assured.

- Final Condition

  A final condition is a condition that once met, stops the mission's execution and may be used as the accomplishment result of the mission.

- Relationship

  A relationship between missions specifies if a given mission contributes or not to the accomplishment of another mission. This creates a dependency between the missions and tasks if needed.

The mKaos studio [15] provides ways to design an SoS from its missions perspective, yet it does not allow the user to implement any of the items, leading to an architectural view of the SoS missions only.

As stated before in Chapter 1, we intend to provide a dynamic model of the system, which would allow the designer to evaluate the viability of an intended mission. From our research, there are mission modeling tools and SoS simulators (such as the MS4Systems [9]) but it lacks an integrated model simulation environment that could provide this analysis from a mission view.

Later in Chapter 3, we will present a tool that performs this environment integration while trying to keep the concept of a mission as close as possible to the one presented in this Section.

2.2.3 piStar GODA

The piStar-GODA [13] is an extension of the tool of the piStar project[11], making it easy to enable new features focused on the cases studied in the research group. The GODA integration itself with the piStar model is a great example of how we can extend the piStar application, for this current work we have used the piStar version available at `https://pistar-goda.herokuapp.com/`.

The piStar tool offers a graphical design platform for designing i* [6] goal models using the *tropos* methodology [4]. A piStar model is shown on the Figure 2.3,

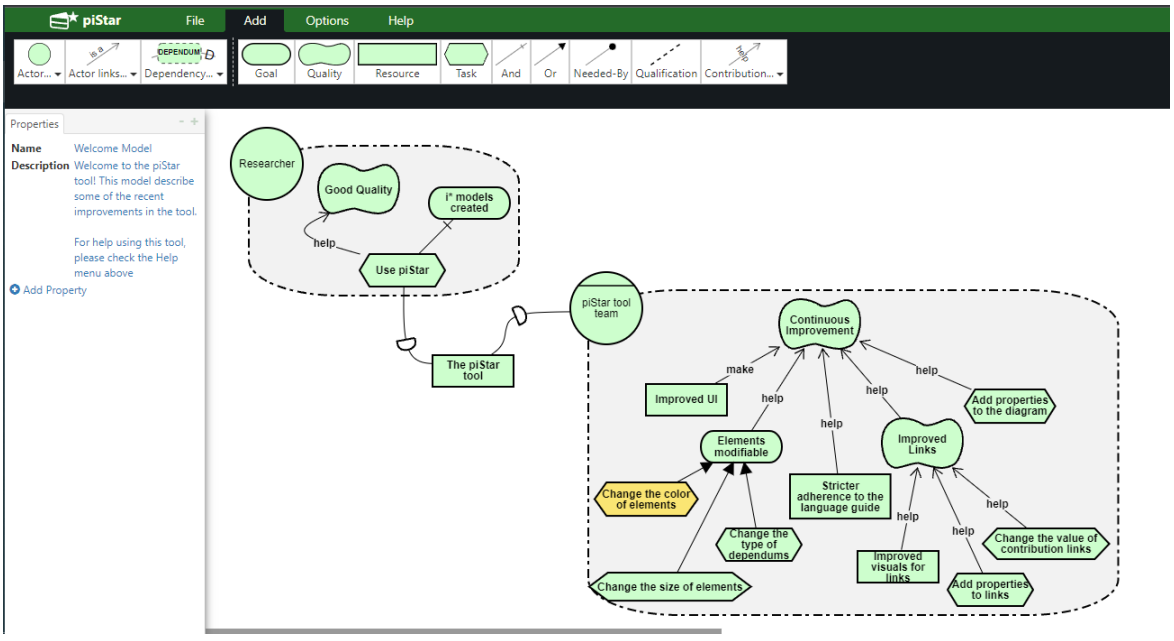

Figure 2.3: A generic piStar model at the platform

This tool allows us to implement goal models regarding the relationships of the elements which is our main concern for an SoS mission. As the mKaos [15] tool presented before, the goals hold the representation for a mission and its decomposition. The model itself is purely declarative too, no implementation is needed at the piStar modeling stage.

2.2.4 SoS modeling on i* Framework

There is currently on the literature some proposals for modeling System-of-systems under the iStar Framework. The work of Alhajhassan[2] proposes an agent based modeling for SoS. Goal-oriented modeling simplifies the design process by elucidating the goals structure for a process, it removes the need of precisely describing agents or systems that participate on the process.

All goal-oriented modeling approaches researched shared common elements, such as agents, goals, relationships and soft goals. The proposed tool will try to follow the same path as stated by the presented studies to be in consonance with the current literature about SoS and SoS modeling.

2.3 System-of-Systems simulation

In this section, we explore the simulation tool for SoS, how it works, and its simulation model.

2.3.1 MS4

Based on the Eclipse IDE, the MS4 Modelling Environment [9] is a platform for SoS simulations, its approach is based on the widely known DEVS (discrete event system specification) [19] framework, which is based on atomic models for the components. The resulting transition system always goes from a previous state to another or else it *passivate*[1] on a defined state, each component has input and output ports for communicating with other systems. Also, it may control as many internal states (variables) as it wants to, and each system is completely independent. So it may respond to many requests from different components of the network.

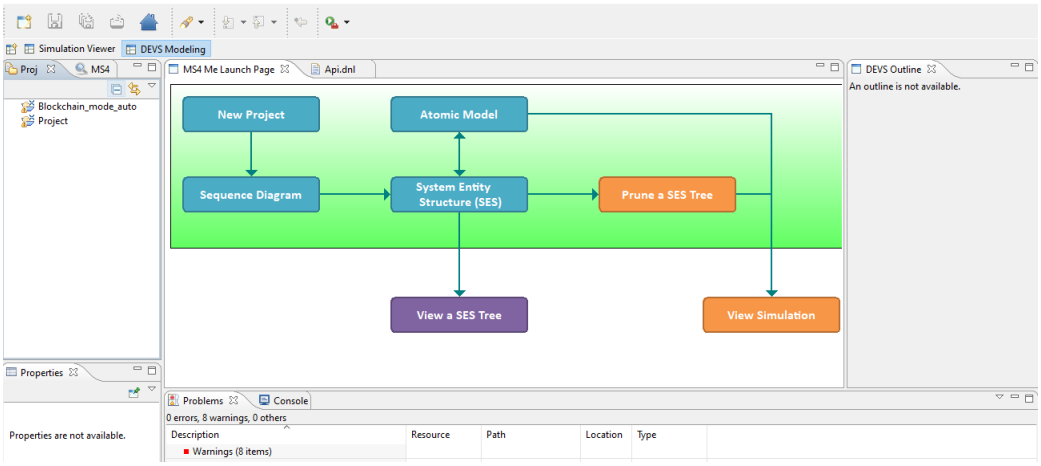

Figure 2.4: The MS4 environment home screen

[1] *passivate*: holds on the state until an external condition triggers it to another state

Figure 2.5: An MS4 component ready for simulation

All the paths are triggered by events emitted by other elements, those emissions may be an autonomous signal - in which case the component has decided for their own reasons that the signal must be sent - or they are responses to external events.

After a component is ready, it can be simulated as an isolated block, and as we can see in the Figure 2.5, it becomes a black box with inputs and outputs so you can test if the system behaviors are being achieved correctly.

The platform works with two input files to create the SoS perspective and to model it: Sequence diagrams and atomic models. For the first, it uses the UML (universal modeling language) [5] specification and keeps the description in XML files that can be imported from other compatible XML based applications, or by design it in the environment itself. Thus it can construct the SoS from an architectural description.

But we want to evaluate an SoS mission execution from a goal model. Then, how to use the MS4 environment to simulate a mission instead of verifying its architecture only? This leads us to the second kind of input: atomic models, which are going to be introduced next in Section 2.3.1.

Once the components are built disregarded of the approach that produced them, it is time to link them and describe their connections to each other. The file responsible for this description is the SES (System Entity Structure) file. It uses a proprietary structured natural language for achieving this. We show an example of it in the Listing 2.1, the SES links the systems (oxygen, hydrogen, 'reactProcess' and water) to simulate water production by the reaction of the elements triggered by the ReactProcess manager component.

```
1 From the reaction perspective, ChemicalReaction is made of
     ReactProcess, Hydrogen, Oxygen, and Water!
2
3 From the reaction perspective, ChemicalReaction sends StartUp to
     ReactProcess!
4
5 From the reaction perspective, Hydrogen sends Release to ReactProcess!
6
```

```
7  From the reaction perspective, Oxygen sends Release to ReactProcess!
8
9  Hydrogen can be LowH or HighH in HydroConcentration!
10
11 Oxygen can be LowO or HighO in OxyConcentration!
12
13 From the reaction perspective, ReactProcess sends ReleaseTwoMolecules
      to Hydrogen!
14
15 From the reaction perspective, ReactProcess sends ReleaseOneMolecule
      to Oxygen!
16
17 From the reaction perspective, ReactProcess sends AcceptOneMolecule
      to Water!
18
19 From the reaction perspective, Water sends MoleculesOfWater to
      ChemicalReaction!
20
21 From the reaction perspective, Hydrogen sends MoleculesOfHydrogen to
      ChemicalReaction!
22
23 From the reaction perspective, Oxygen sends MoleculesOfOxygen to
      ChemicalReaction!
```

Listing 2.1: SES file example for an SoS

The SES file is then converted into a specialization tree - shown in the Figure 2.6 -, and at this stage we can specialize those elements that have variants (this is expressed by the "*{ComponentType} can be X or Y in {Unspecialization}*" statement, at line 9 and 11 in the Listing 2.1).

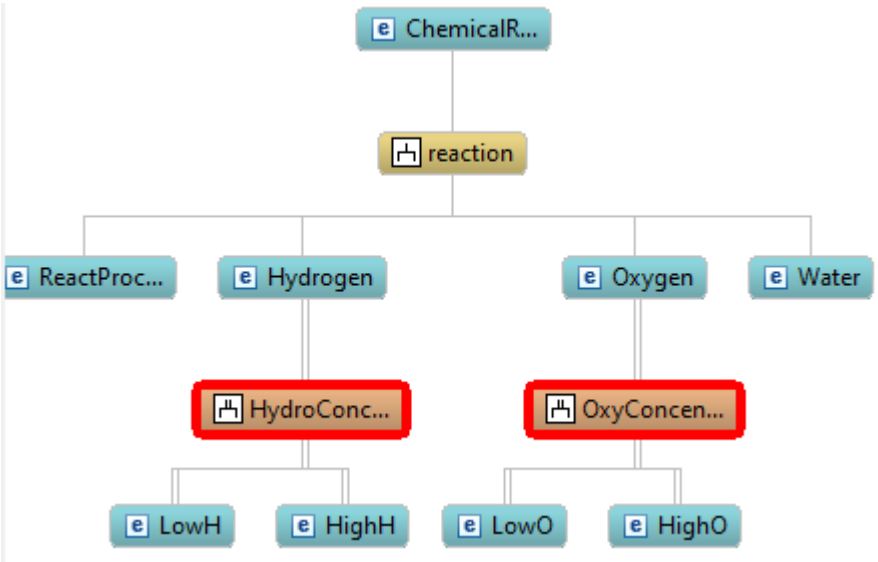

Figure 2.6: SES tree view

Once it is pruned, the SoS is ready for simulation, the resulting view of the mesh can be seen in Figure 2.7, the lines represent the connections between the components. Note that they only link output ports to input ports. Everything that happens inside a component is unknown from the SoS perspective, which corresponds to the expected for the SoS abstraction seen before in Section 2.1. Often, the designers may not know how those components are implemented, but they have in hands those subsystems behaviors (behavior here means: "for a given input the system outputs an expected value (or range) or else it emits a generic signal") well-mapped through API documentation, structural design, behavioral tests, etc.

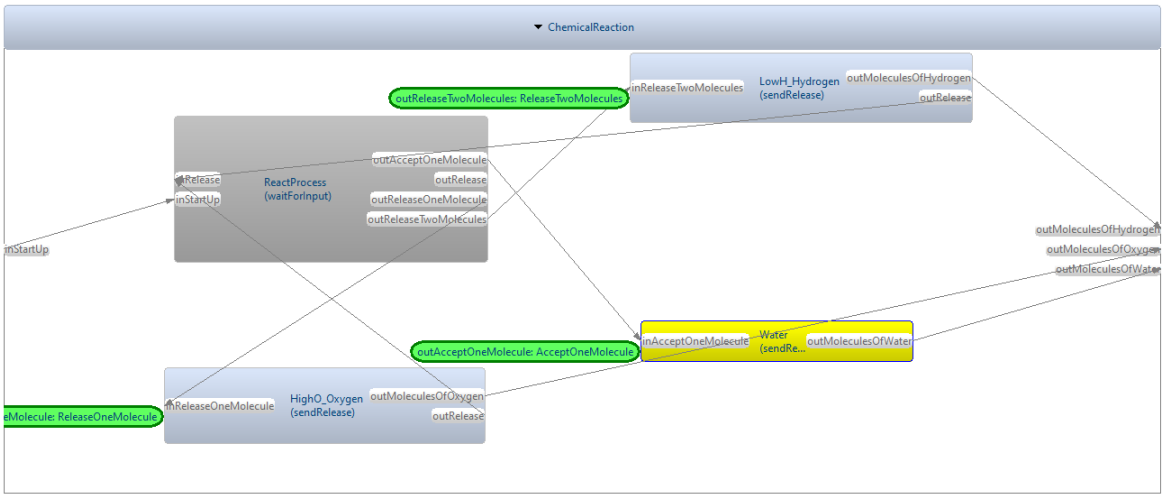

Figure 2.7: React process SoS simulation running

Atomic models

Another way to model the MS4's components is the atomic models, as referred early in this section. Those are files written in another yet structured language: DNL (DEVS Natural Language) [10]. Bellow we show the ReactProcess DNL file from the previous subsection as an example:

```
accepts input on StartUp!
accepts input on Release with type IntEnt!
use Release with type IntEnt!
generates output on AcceptOneMolecule !
generates output on ReleaseTwoMolecules!
generates output on ReleaseOneMolecule!
use releaseTime with type double and default "0"!
to start hold in sendRelease for time "releaseTime" !
//
passivate in waitForInput!
when in waitForInput and receive StartUp go to sendRelease!
when in waitForInput and receive Release go to sendRelease!

hold in sendRelease for time "releaseTime" !
```

```
from sendRelease go to waitForInput!
after sendRelease output Release!
external event for waitForInput with Release
<%
    IntEnt Release = null;
    for (Message<IntEnt> msg : messageList) {
        Release = msg.getData();
        if (Release.getValue() == -1) {
            break;
        }
    }
    int income = Release.getValue();
    if (income >0){
        releaseTime = 1;
    }
    else{
        releaseTime = Double.POSITIVE_INFINITY;
    }
%>!
output event for sendRelease
<%
    output.add(outReleaseTwoMolecules, "ReleaseTwoMolecules");
    output.add(outReleaseOneMolecule, "ReleaseOneMolecule");
    output.add(outAcceptOneMolecule, "AcceptOneMolecule");
%>!
```

Listing 2.2: A DNL file example

The DNL file can be extended on its transitions and events by adding Java code that will perform actions on the state of the component, process the input, or add a value to an output signal (i.e. it outputs a custom value for the signal, allowing the system not only to trigger other systems but indeed to *communicate* with others in the network).

By knowing its structure we can generate it by meta-programming the component, as its base is a natural language, the system's behaviors can be easily understood by anyone.

Also, the capability of writing Java code directly into the transitions supports creating components that more than follow state paths also do calculate something - by interfacing with some real-world system or by mocking values - and verify conditions of interest for the SoS implementation.

With those features (DNL, SES, and Java extension code) we may achieve our goal of modeling missions for an SoS. In Chapter 3, we propose a mapper capable of converting a goal-oriented mission model to an SoS perspective ready to be simulated on the MS4 Modeling Environment.

2.4 Blockchain as a Cyber-physical SoS

As stated by other works [16], a blockchain may be seen as SoS since it is formed by autonomous agents that operate over the same set of data. The emergent behaviors that can surge then are infinite. In this work it will be used a blockchain network as the domain of the mission modeling.

Van Lier [16] focuses on the applications of blockchain and the emergent behaviors that derive from sharing a piece of data, i.e. a network between a supermarket and its vendors may be able to optimize the delivery of the products and to improve predictions about the consumer demand, making the supply chain faster than before.

Instead, this work will aim in general missions of blockchains, such as the registering process, integrity check (which is performed as a sum of contributions from each agent in order to validate the good faith of the network participants) and so on.

It may be possible to model blockchain contracts using the proposed model in this manuscript, but the perspective about the model agents must change: instead of seeing the network elements and their behaviors, the model must design the contribution of data from each network participant, i.e., In the previous example how the consumer and the vendor behaviors with regard to the context of the data that they share, for example the available stock amount. Following works may try to validate this hypothesis since it was taken out of the scope of this project due to time limitations.

Blockchain Elements (Constituent Systems)

A private blockchain network is formed by four independent systems that collaborate together in other to assure some expected properties from the blockchain, such as integrity, real-time synchronization, block validation, etc.

The four major elements that constitutes a blockchain network are:

- **Peers**: responsible for executing the smart-contract transaction and holding the current state of the world;
- **Orderers**: servers responsible for the validation of the block as for to keep their sequence immutable;
- **Chaincode**: also known as smart-contract, it is a piece of code instantiated on the network which is called at every transaction. Responsible for validation and calculations for a transaction running on the blockchain; and
- **SDK**: any application that is capable of communicating to the blockchain network through an SDK, it may be a phone app, web app, etc.

Capabilities

Some capabilities on this proposed blockchain are assumed for the constituent systems of this blockchain, defining what are the assumed capabilities is important

for the mission modeling that will be presented later in Section 3.2.1, those skills are:

- An SDK application is able to communicate to the peers through gRPC and REST protocols. Additionally, it can broadcast search messages on the network for all peers, using either TCP or UDP protocol.
- A Peer is able to broadcast its result to all the applications listening for transactions or to unicast the result to the last online application.
- An Orderer will notify the network through broadcast every time that a new block is added to the chain.

Chapter 3

Proposal

In Chapter 2, the problem was made clear: there are goal-oriented approaches for SoS modeling (Section 2.2) and there are simulation tools for SoS (Section 2.3), but there isn't any tool capable of integrating these two methods.

The problem that remains is: How to model an SoS from a mission (or goal) perspective and simulate it in the terms of its behaviors? This is what the present work tries to solve with the proposed tool presented in Section 3.2.

In Section 5.1, it is shown how to download, install and execute the GM2MS4 tool, with an exported piStar model as the input.

3.1 From goal-oriented model to MS4 model

When converting from a piStar model to an MS4 model it is important to know how the elements from one are associated to the other. In this section it will be explained how the elements are related, starting at the conceptual model for an SoS mission and the piStar elements and in the sequence the relation between the iStar and the MS4 ME.

Using the mKaos conceptual model for a mission given in Figure 2.1, its elements are represented both in the piStar and MS4 as shown in the Table 3.1.

mKaos	piStar Model	MS4 Model
Mission	Each mission (individual, global or common) is represented as a goal with a property that sets the constitutent system where this goal must be achieved	Each goal under the tree becomes a linked state, having a predecessor and a successor state, the state sequence is extracted from the tree structure (goals that reside on higher levels are executed before the ones in lower levels of the tree)

Table 3.1 continued from previous page

mKaos	piStar Model	MS4 Model
Constraints	AND/OR links are capable of restricting the mission execution in case of failure.	At each state transition the runner method verifies the last execution result and evaluates the continuation of the mission based on the relation of that state (goal) to its parent. This evaluation decides if the system will do its calculations and contribute to the mission accomplishment or not.
Task	Tasks are represented with iStar elements of the same name.	Each task is represented by a Java method that receives the last execution result and returns the next result
Task parameter	Not present in the model	Not present in the MS4 model
Priority	Priorities can be set in the order array specified along the goal name. i.e., [T0;T3;T2] prioritizes the tasks T0 and T3.	The order that the tasks methods will be called during a state transition
Connectivity	The model allows connectivities to be expressed by linking goals with different "component" properties.	Connections are abstracted into signal events from a constituent system to another, these signals when emitted carry the goal execution result. This allows each system to evaluate if they must execute their tasks related to that requisition or not, in simpler words: Given the last state of the mission, should the system colaborate with it?

Table 3.1 continued from previous page

mKaos	piStar Model	MS4 Model
Executor	The root node for a tree of goals representing a mission, also called verifier on this text, this is the system responsible for executing the mission and for its result evaluation	An aditional constituent system is created to validate the mission final result. The execution rules and signal behaviors are the same as the ones from the others contituent systems.
SoS	The set of goals of the distinct systems that are part of the SoS. The tree present in the agent on the model.	The set of connections of distinct systems that participates on the SoS. Also called SES file, as presented before.

Table 3.1: Relation between the models elements

3.2 piStar-MS4 goal mapper

While the goal-model exported file has its own definitions[12] and represents only the SoS goals, the simulation comes from the MS4 model which describes systems transitions of the constituent systems, or **behaviors**: the DNL files. Linking these DNLs together in a SES file will enable the SoS simulation.

The proposed tool is named GM2MS4 (an acronym from "Goal Model to MS4") and will be referred to as such from now on. It is responsible for the mapping from the model to the simulation files, making it possible to simulate the goals of an SoS from its mission model exported by the piStar-GODA [12].

The next sections will show how this was accomplished and how to run the software. Starting from the model construction (Section 3.2.1) where we show how to construct a mission model in the piStar-GODA that is compatible with the GM2MS4 tool.

The implementation in Section 4 presents how the tool performs the transformation from the proposed model to the simulation project.

3.2.1 Model construction

As mentioned before, the models are constructed in the piStar-GODA tool[1]. Our model represents a mission for an SoS, and the mission is represented in the application as an agent containing a goal at the root level of the agent tree. The children of the root goal are then a composition of other missions that must be accomplished by other systems.

[1] available at `https://pistar-goda.herokuapp.com/#`.

The model uses three elements from the modelling application:

- Goals
  - Represents a high-level abstraction for an expected behavior, it constitutes a mission in our model;
  - Each goal must have a property named "component" in the model indicating what is the constituent system that must accomplish it;
  - It can be decomposed into other goals, and those may or may not have the same property *component*.
  - It can be decomposed into tasks[2];
  - The goal name must start with a unique identifier in the form of "G%d:", where %d is any natural number, for example: G12, G5;
  - The goal name must end with the order of execution of its children, separated by a ';' and inside square brackets, the order is then defined from left to right . i.e. G0: A Super Goal [T1;T2;G1] is a valid name for a goal, defining the execution order as T1 -> T2 -> G1;
  - In the execution order array, the tasks must always precede other goals.

  In Figure 3.1 we show a simple model for an SoS mission. Each color designates a different system where the goal must be achieved. The legend in Figure 3.1 shows what constituent system the color represents in the model[3]. The color code will be kept as shown in the Legend 3.1 for the examples presented next.

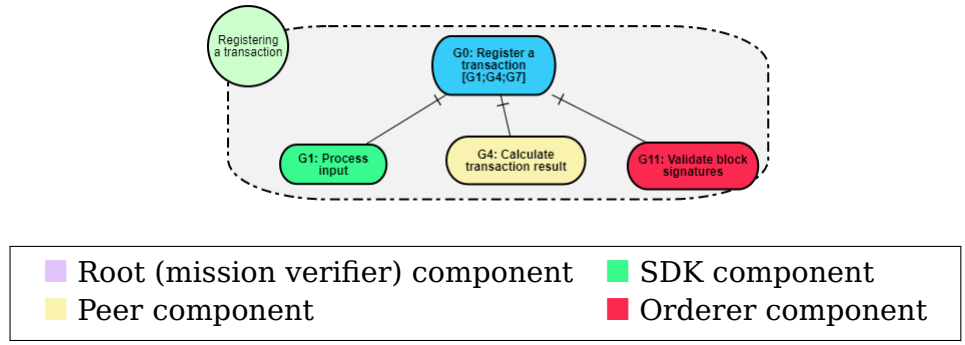

Figure 3.1: A simple mission model containing only high-level goals

- Task
  - Represents an implementation method for a constituent system process;
  - Each task must have a property named "component" in the model indicating what is the constituent system that must accomplish it;

[2]note that it does not make sense to execute a task in a different system to achieve the goal of some other system. So a task must belong to the same system as its parent.

[3]the colors are for understanding purposes only, they do not represent anything about the model conversion. The constituent system name must be set at the "component" property at the goal element.

- All the leaf tasks on the model tree will be converted into Java methods. These methods together define the necessary interface for the components to communicate while respecting the relation provided in the model;
- A task must always relate to a parent of the same component type;
- A task may be decomposed in other tasks and, in this case, they are named *refiner* tasks;
- A task must not be decomposed into goals;
- The task name must start with a unique identifier in the form of "T%d:", where %d is any natural number;
- The task name must end with the order of execution of its children, separated by a ';' and in between square brackets, the order is then defined from left to right. i.e. T0: A Super Task [T3;T4;T5] is a valid name for a task, and it defines the execution order as T3 -> T4 -> T5.

This element allows us to interface with the real world or simulate processes through the Java programming extension provided by the DNL file.

In Figure 3.2, the refinement of a task is shown, and the leaf tasks are converted later into Java methods.

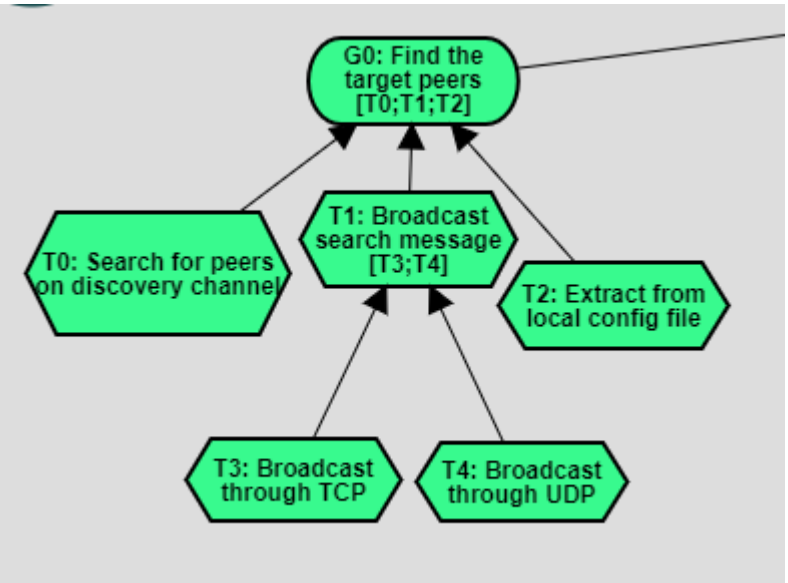

Figure 3.2: The task T1 is refined into T3 and T4

- Links
  - AND links: represents a dependency between the children. In other words, the first child must be accomplished before the second starts. In Figure 3.3, it is shown an example for a goal having two children linked by an AND relation.

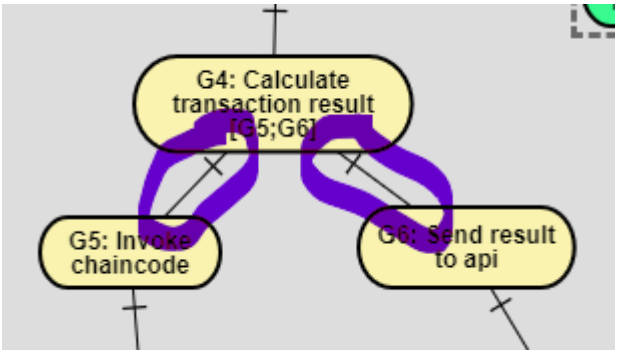

Figure 3.3: An AND link where both G5 and G6 must be accomplished

- OR links: Represents a *variability* between the children where the accomplishment of only one of the children is necessary to achieve the accomplishment of the parent. In Figure 3.4, a variability example is shown, where the success of any of the tasks accomplishes the goal G6.

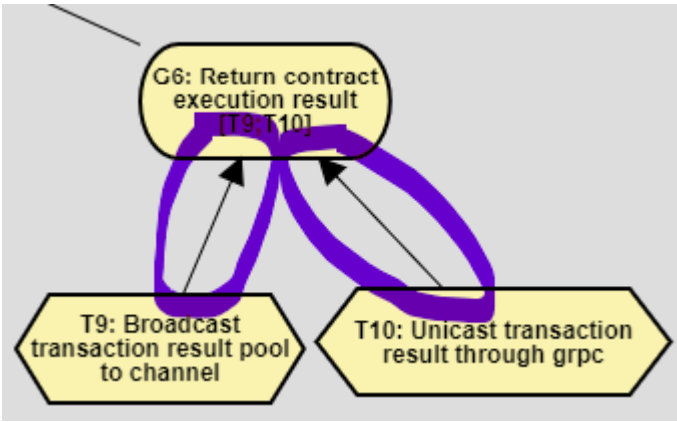

Figure 3.4: An OR link: any of the tasks must be accomplished

Links can be used to decompose a task - creating a refined task - or to decompose a goal into other goals and/or tasks. Figure 3.5 shows a goal decomposed into a task and another goal. In this case, all tasks must precede the goals in the execution order. Graphically that means all the tasks must be placed at the left of the goal children of the parent.

The number of children of a parent is unlimited per the model specifications but they must all be connected with the same type of link.

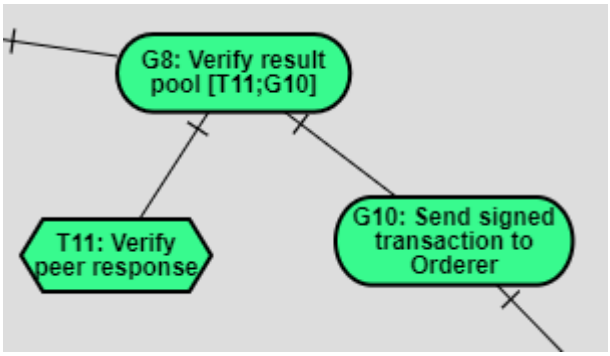

Figure 3.5: A goal decomposed into a task and another goal

An SoS model example using Blockchain

In Figure 3.6, a complete goal model for a blockchain network is shown. The next Section shows how this model is converted into an MS4 project.

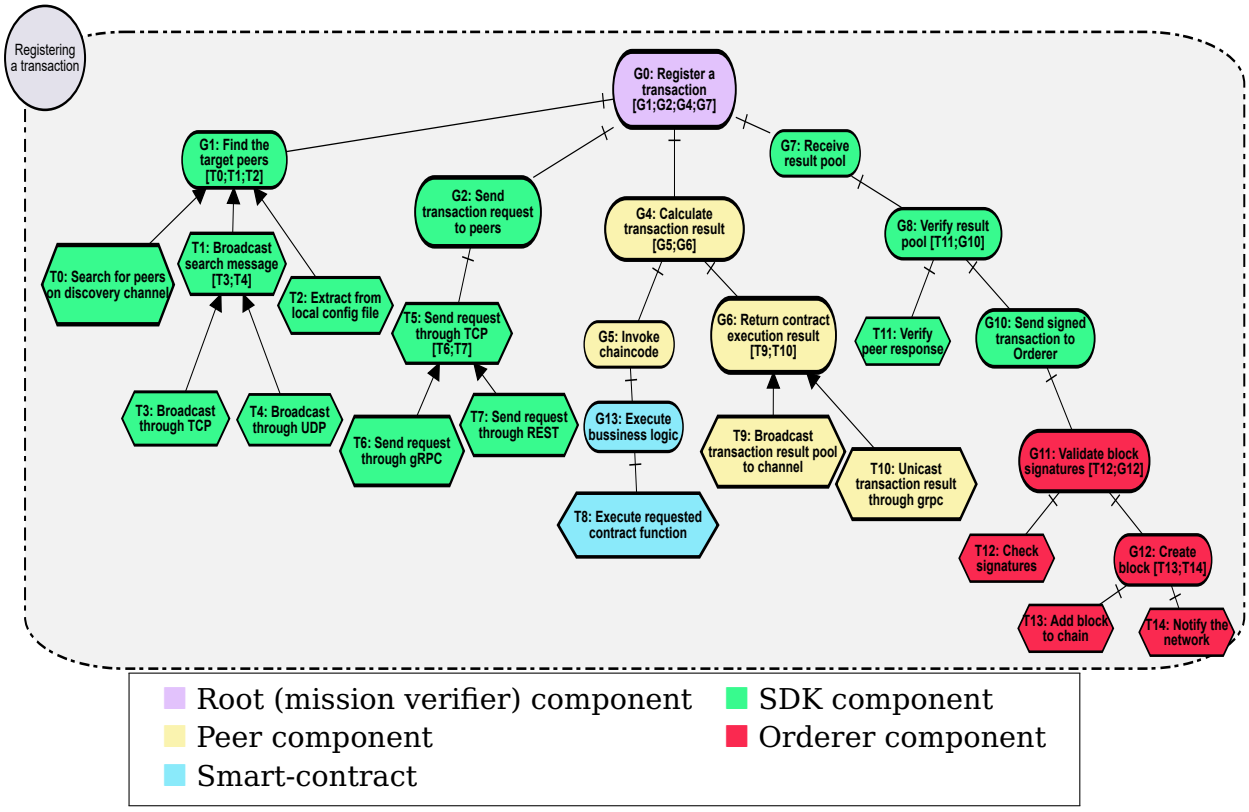

Figure 3.6: A complete example for the proposed blockchain

Chapter 4

Implementation

In the topics ahead, the mapper implementation is presented in the following structure:

- Model conversion: Model loading and parsing process. It is shown how the piStar model is converted into the main structure used by the map which is the GoalTree;
- Connections: Definition of the structure of connections. What are the model rules that describes an interconnection between different systems and how the structure is calculated;
- DNL file generation: Shows the functions that are responsible for generating the DNL, naming rules for ports, and how the state sequence is generated from the goal model. Also, it is explained how a system triggers another one and waits for their response;
- Generation of Java classes: explains which Java classes are created along with the DNL file and how they contribute to the simulation process, and
- SES file generation: points out which previous structures are used in order to create the SES graph.

The GM2MS4 tool was written in Typescript, which is a superset of the JavaScript language that allows typing for the objects and structures. All the types of structures presented in the listings of this Section are in Typescript type notation.

To simplify reading comprehension, the function prototypes presented in this Section follow the notation shown in Figure 4.1.

functionName(parameter: parameterType) $\Longrightarrow$ **ReturnType**

Figure 4.1: Function notation

4.1 Model conversion

Once the piStar [13] model is ready, it can be exported as a JSON file by clicking at "File > Save Model" option from the toolbar. The saved model will be used as the input for the GM2MS4. The exported JSON file for the example shown in Figure 3.6 can be found in Appendix I.

After the model isLoaded, it is validated by the function **validateModel(model: Model) => void**, responsible to check if there is more than one root in the model. If so, the model is taken as invalid and the conversion stops.

The component property of the root goal isn't restricted and the designer may set the value wished for naming the mission verifier module, it won't make a difference for the model processing. As stated before, this node is responsible for evaluating the result of the mission execution. Once the mission has finished it will output a fail or pass for the SoS simulation.

Then, the function **convertToTree(model: Model) => GoalTree** converts the loaded model into a tree that can be represented by the recursive type described in Listing 4.1. This function starts from the root node (the one with the property "selected: true") and then searches for its children in the link array. This process is repeated recursively for each child until it reaches the leaf nodes.

```
1  interface GoalTree {
2      // node data
3      component: string
4      type: 'task' | 'goal' | 'resource'
5      isRoot: boolean
6      identifier: string
7      text: string
8
9      // children and relationship data
10     relation:  'and' | 'or' | 'none'
11     children?: GoalTree[]
12 }
13 // type alias
14 interface Node = GoalTree
```

Listing 4.1: Tree type returned after model conversion

At the end of the calculation, the model is converted into a *GoalTree*, which is the main structure that is operated in the tool, from which the desired properties are extracted and the project for MS4 is created.

In line 14 of Listing 4.1 a type alias called Node is defined to refer to the GoalTree. Sometimes a function may return a Node array to indicate that the original tree was decomposed in many other trees containing only the desired elements.

4.2 Connections

Before creating the MS4 model, it is necessary to navigate the goal tree to search for connections between different constituent systems. These will be necessary to describe the ports of a component (input and output) in the MS4 representation.

The function **componentConnections(tree: GoalTree) => Connections** applies a recursive reduce on the tree, searching for goals linked to goals from a different component. Every child with the property "component" distinct from its parent is then added to the Connections structure, adding an output port for the parent and an input port for that child.

The Connections structure is presented below in the Listing 4.2:

```
type port = {
    inputPortName: string
    outputPortName: string
    from: connectionNode
    to: connectionNode
    rootLink: boolean
}
type Connections = {
    [K: string]: port[]
}
```

Listing 4.2: Connnections structure

For a tuple $(from : Node, to : Node)$, the properties "inputPortName" and "outputPortName" are named based on the rule presented in Listing 4.3:

```
inputPortName: `From_${to.identifier}_to_${from.identifier}`,
outputPortName: `From_${from.identifier}_to_${to.identifier}`,
```

Listing 4.3: Ports naming

With this dependencies, the SoS constituent systems will be capable of communicating with each other, adding dependency between them, which means: a system may depend on another system to accomplish some goal.

Every time a component sends a signal to another system, it returns to its initial state where it will wait for the next event.

Once the triggered goal on the targeted system is accomplished (or not), the called system will trigger the caller back by emitting an event on its respective input with the response for the caller request.

Example: If in the model, component A is linked to component B through the goals G1 and G2, during simulation, component A must trigger component B at input port "From_G1_to_G2" using its output port with the same name. Once this branch execution is finished, component B will trigger component A back on input port "From_G2_to_G1" using an output port with the same name. The signal carries the goal execution result. This communication is better represented in Figure 5.11.

4.3 DNL file generation

For the DNL writing, first, we group the root branches by component, the function

getTreeNodeByComponent(nodeType: goal | task, tree: GoalTree) =>
[component: string, rootGoals: Node[]]

results in a tuple containing the component name ("component" property set in the goal model) and its input goals. Input goals are defined as the goals which are connected to a parent having a different component property. If the goal is linked to a parent of the same component type then it is taken as the successor state in the MS4 model.

All components have the same initial state: waitForInput.

Taking the third branch from the example shown in Figure 3.6, it is extracted the following state sequence in the MS4 model for the Peer component:

$$InputFrom\_G0\_to\_G4 \rightarrow [waitForInput \rightarrow G4 \rightarrow G5] \rightarrow OutputFrom\_G5\_to\_G6$$
$$InputFrom\_G6\_to\_G5 \rightarrow [waitForInput \rightarrow G5 \rightarrow G6] \rightarrow OutputFrom\_G4\_to\_G0$$

Note that G6 outputs at the "From_G4_to_G0" port, this is a rule: the last goal of a component on the branch outputs back at the output port associated with the root node of the sequence.

Every time a component outputs a signal it must wait for the response of that system. This is done by making it go back to the initial state "waitForInput". As the DNL model is atomic, no conditional branch between states is allowed. To allow the system to continue the dependent process, an auxiliary state was added to hold the transition after an external component has replied with the requested goal response. The state is named by the following rule:

$$[CallerIdentifer]\_continue \tag{4.1}$$

where [CallerIdentifier] is the identifier tag of a goal in the set of output goals.

I.e., for a set [G0, G2, G6, ...] representing the states which trigger a system of a different type of a component, a set of the same size containing the states [G0_continue, G2_continue, G6_continue, ...] will be created to resume the tree execution at the point which the component went back to its initial state.

The generated DNL file for the peer from the example model (Figure 3.6), is available in the Appendix II.

4.4 Java class generation

As pointed out before in Section 2.3.1, the MS4 DNL files can be extended using Java code compatible with release 1.6 of the language (the MS4 environment runs under this version).

Each component defined in the model leads to two classes: {component}TranstionsClass and {component}TaskClass. i.e, for a component named SDK, two classes are generated: SdkTransitionsClass and SdkTaskClass.

The transition classes define methods terminated by "_runner" which are called at each state transition of a system. Those methods will call methods defined in the corresponding task class - for the task class, the methods end with "_task" - arranging their sequence execution and verifying if the result corresponds to the relation expected for a goal to be achieved. The call of the runner method can be seen in line 58 of the listing II.1 in Appendix II.

```
1  public Result return_contract_execution_result_runner(Result result) {
2
3      result = verifyContinuation(result, "and" , true);
4      if (result.locked()) {
5          return result;
6      }
7
8      TaskRunner[] runners = new TaskRunner[] {
9         new TaskRunner() {
10         public Result run(Result res) {
11          return PeerRunner.Broadcast_transaction
12                          _result_pool_to_channel_task(res);
13         }
14        },
15        new TaskRunner() {
16         public Result run(Result res) {
17           return PeerRunner.Unicast_transaction
18                          _result_through_grpc_task(res);
19         }
20        }
21     };
22     return tasksRunner(runners, "or", "and", result);
23     //Goes to state: output_state
24 }
```

Listing 4.4: PeerTranstionsClass.java

In Listing 4.4 the runner method for the goal G6 is shown. First, it verifies if it can continue based on the last result and its relation to the parent and then groups the tasks in an array to be executed by the *tasksRunner* method.

The method *verifyContinuation* will lock the result state of the component if a child linked by an OR link executes successfully (there is no need to process any new tasks at the same level). If a task linked by an AND link fails then the following tasks should not be executed.

The *tasksRunner* method is responsible to execute the runner sequence, evaluating at the end of each task execution if it may continue or not.

The *PeerTransitionsClass* and *PeerTaskClass* can be found in appendices III.1 and IV.1

Two auxiliary classes are provided by the tool: *Result* and *Error*. The first holds the result that is carried through transitions, where each component has a variable of type Result to hold the result state of the last executed task. The second holds an Error value inside the Result state and the user may modify the class in order to add more info to the result in the case of an execution error. These classes are available in Appendices V.1 and VI.1

4.5 SES file generation

As the last step of the transformation, the SES file is generated. The output and input ports are matched from the arrangement of the Connections structure presented before, with an array of goals for each component (both structures are calculated in the first step of the conversion. Those structures are the information needed to create the connections graph expressed in the SES file. An SES file example for the model presented in this section is given in the Listing 4.5.

```
1  from the blockchain perspective, Registering_a_transaction is made of
       Verifier, Peer, Chaincode, Sdk, and Orderer!
2
3  from the blockchain perspective, Registering_a_transaction sends
       StartUp to Verifier!
4
5
6  from the blockchain perspective, Peer sends From_G5_to_G13 to
       Chaincode!
7  from the blockchain perspective, Chaincode sends From_G13_to_G5 to
       Peer!
8  from the blockchain perspective, Verifier sends From_G0_to_G4 to Peer!
9  from the blockchain perspective, Peer sends From_G4_to_G0 to Verifier!
10 from the blockchain perspective, Verifier sends From_G0_to_G7 to Sdk!
11 from the blockchain perspective, Sdk sends From_G7_to_G0 to Verifier!
12 from the blockchain perspective, Verifier sends From_G0_to_G1 to Sdk!
13 from the blockchain perspective, Sdk sends From_G1_to_G0 to Verifier!
14 from the blockchain perspective, Verifier sends From_G0_to_G2 to Sdk!
15 from the blockchain perspective, Sdk sends From_G2_to_G0 to Verifier!
16 from the blockchain perspective, Sdk sends From_G10_to_G11 to Orderer!
17 from the blockchain perspective, Orderer sends From_G11_to_G10 to Sdk!
18
19 from the blockchain perspective, Verifier sends stop to Peer!
20 from the blockchain perspective, Verifier sends stop to Chaincode!
21 from the blockchain perspective, Verifier sends stop to Sdk!
22 from the blockchain perspective, Verifier sends stop to Orderer!
```

Listing 4.5: SES file for the example model

Chapter 5

Execution

In this chapter it is presented how to run an SoS simulation using the project that the GM2MS4 tool exports. The sections will show how to:

1. Execute the GM2MS4 program
2. Load the project into the MS4 ME
3. Create execution conditions for the leaf tasks
4. Evaluate the simulation result

5.1 Running the converter

The tool may be run from a development environment or directly from the npm package. In the first case, the user must download the code from the GitHub repository[17] (available at `https://github.com/vieirin/GM2MS4`). If they want to use a different model, the model path that parametrizes the *loadModel* function must be changed to the path of the new model first. Then the tool can be executed by running the command given in Figure 5.2. For the second case, no repository download is needed and the process is explained at Section 5.3.

In the following sections we present the project dependencies, how to execute the tool, how to import the outputted MS4 project inside the MS4 ME, and finally how to simulate the mission.

5.2 Dependencies

To run the converter, the user must have Node.js installed on the machine.

In Listing 5.1, the libraries that the tool depends on to run are shown. They can be installed by invoking the command *npm install* from the root directory of the repository.

```
1    {
2        ...,
3        dependencies: {
```

```
            "@types/node": "^12.11.5",
            "jszip": "^3.7.1",
            "lodash.intersectionby": "^4.7.0",
            "lodash.merge": "^4.6.2",
            "lodash.mergewith": "^4.6.2",
            "lodash.uniqby": "^4.7.0",
            "ora": "^4.0.2",
            "replace-in-files": "^2.0.3",
            "typescript": "^4.3.5",
            "yargs": "^14.2.0"
        },
        ...,
    }
```

Listing 5.1: Dependencies Sectionfrom the package.json file

5.3 Execution

The converter binary is available at the npm registry "gm2ms4-dev" (`https://www.npmjs.com/package/gm2ms4-dev`) and can be run using the npx command from Node.js installation folder. Npx will download the package and save its binary into the bin directory of Node.js. The command for running using npx is shown in Figure 5.1

npx gm2ms4-dev - -output=<output-name> <model-name.txt>

Figure 5.1: Npx command line for running the conversion tool

Additionally, the user can execute the project from the downloaded code, the command is present at 5.2.

npm install && npm start

Figure 5.2: Npm command for running the conversion tool

5.4 Importing onto the MS4 ME

After running the tool, a *.zip* file is saved onto the disk (the name of the zip file defaults to **Project.zip**, but if the tool was ran with the command provided at the Figure 5.1, the user may rename it to the desired name using the flag – *output*). This is the asset needed to import the project into the MS4 ME workspace.

A tutorial for the importing process is given below:

1. Right-click on the workspace area and then select the "Import..." option as shown in the figure

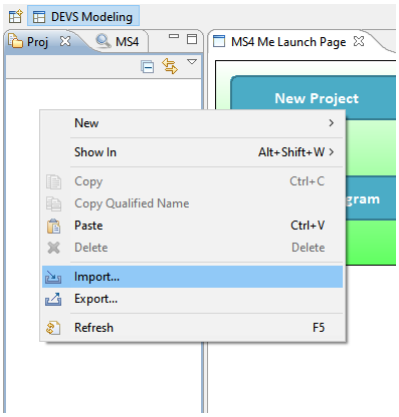

Figure 5.3: Opening the import window

2. In opened window, select the "Existing Projects into Workspace" and then hit the "Next >" button

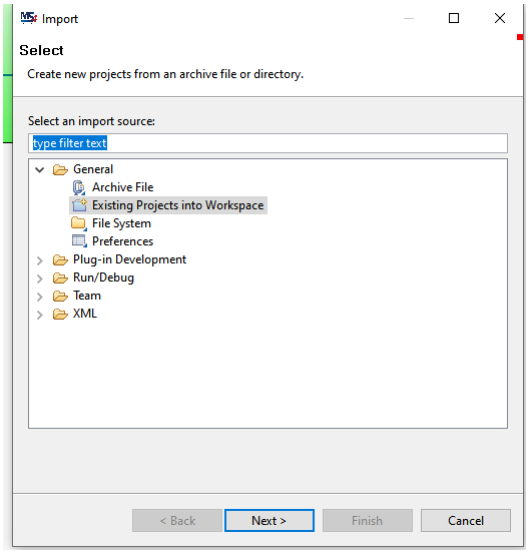

Figure 5.4: Source type selection

3. In the next section, select the generated project .zip file. A project must appear under the window "Projects" area as shown in Figure 5.5, make sure it is selected and hit the "Finish" button.

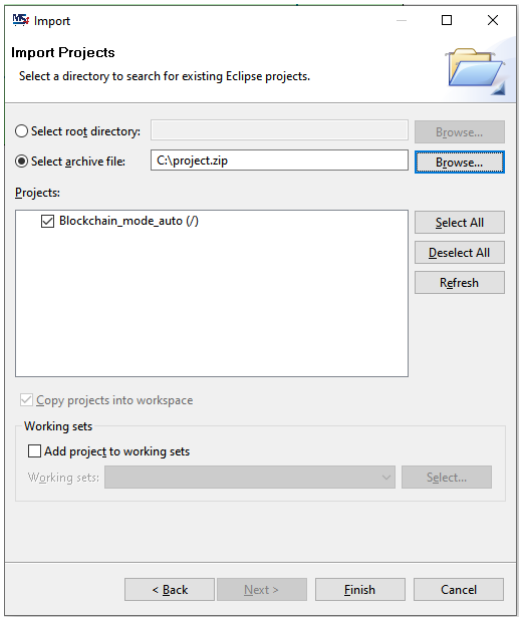

Figure 5.5: Project import window

4. Once the project is imported into the workspace, it must be rebuilt. By rebuilding the project MS4 will generate automatically the Java files for the DNLs that were created by the GM2MS4 tool, and those are the files interpreted by the simulator. Go to the toolbar of the MS4 ME and select "Build > Rebuild Project". In the window that appears next hit the "OK" button.

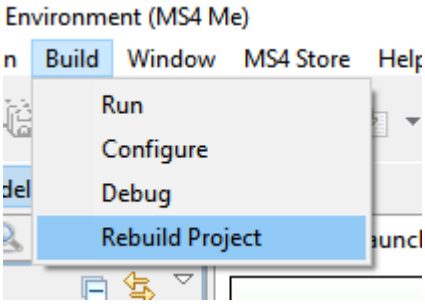

Figure 5.6: Rebuild project

5.5 Before running the simulation

With the project imported into the workspace, the user must now implement the tasks methods for each of the components. All task methods must return a Result that is successful or not. Any calculation can be done in the method but it must always return a Result instance so that the Runner class that called it knows if the tree execution may continue or not.

An example is shown for the *peerTaskClass.java* in the Listing 5.2. The reader may notice that the result depends on the random result, i.e., this is a simplification

to show that a task may succeed or fail based on an external result. The random line could be an HTTP call to a service or even a method calling to another user-defined class, and the possibilities here are infinite. For the model checking though, only the result of the calculation is used, so the user must set the received result to Error or Success before returning it.

```java
package components;

import java.lang.Math;

public class peerTaskClass {

    public Result Broadcast_transaction_result_pool_to_channel_task
       (Result result) {
        // the following line may can be an HTTP call or any process
           you want
        int success = (int)(Math.random()*10);
        if (success < 5) {
            result.setError("The task has failed");
        }else {
            result.setSuccess("The task has succeeded");
        }
        return result;
    }

    public Result Unicast_transaction_result_through_grpc_task
       (Result result) {
        int success = (int)(Math.random()*10);
        if (success > 5) {
            result.setError("The task has failed");
        } else {
            result.setSuccess("The task has succeeded");
        }

        return result;
    }
}
```

Listing 5.2: Implemented peerTaskClass.java

If any of the methods *setError* or *setSuccess* are not called then the result will repeat its last state. Every time an event is triggered in a component the success field of the Result instance is reset to its default, which is False. So the user must set the conditions for the tasks to succeed in order to make the branch pass.

This abstraction allows the designer to evaluate the emergent behaviors of the mission. Sometimes a mission simply cannot be achieved because some capabilities are missing for the designed system. Or on the contrary, the designer can find similar capabilities between the components of the mission. By pointing these out,

the mission model can be modified to either remove a redundant task or to add a new variability to a goal since a common capability was found across the systems.

5.6 Running the simulation

Once all the tasks are implemented the project is ready to be simulated. To do so a tutorial is provided below:

1. Open the *.ses* file living under the "Models.ses" directory and then hit the "Prune SES into PES" button above the file as shown in Figure 5.7

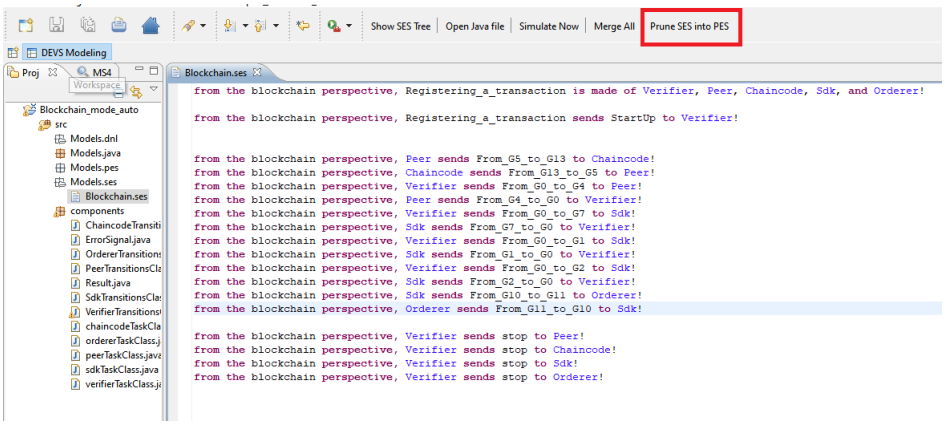

Figure 5.7: Pruning the SES file

2. The MS4 ME will open the PES file automatically. Once that happened, hit the "Run PES file in SimViewer" button indicated in Figure 5.8

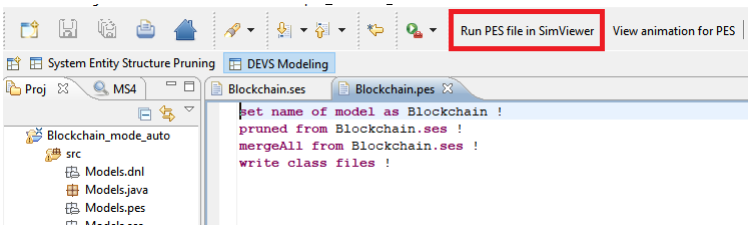

Figure 5.8: Opening the PES file in the simulator

3. Right-click the "inStartUp" port and select the inject input option that appears and then hit the "Run" button at the bottom toolbar.

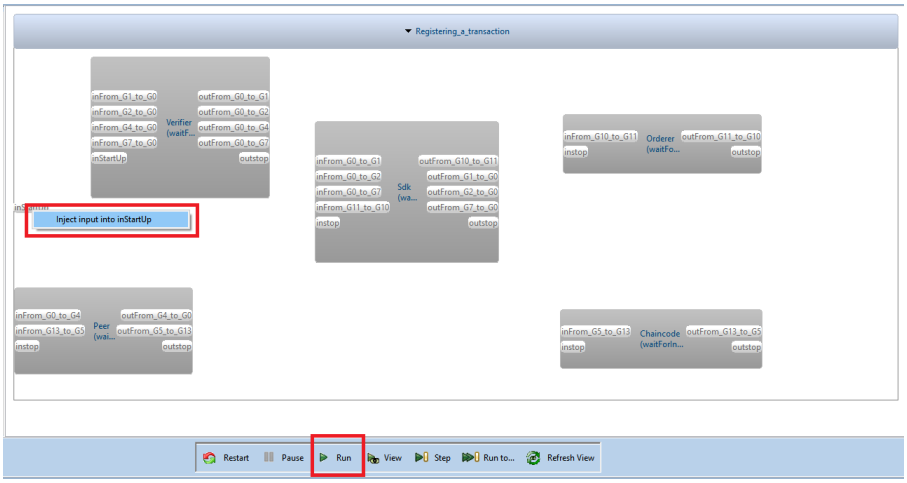

Figure 5.9: Starting the simulation

4. When finished, the simulation opens a dialog window indicating the mission result as shown in Figure 5.10.

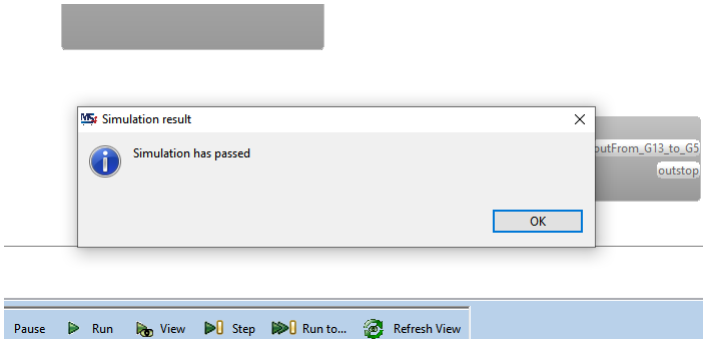

Figure 5.10: Simulation result

The code for a task class is present at the Listing 5.3, by explicitly calling the method **setSuccess** or **setError** in the task, it is possible to force the task into the desired state.

```
1  package components;
2
3  public class ordererTaskClass {
4
5      public Result Check_signatures_task(Result result) {
6          result.setSuccess();
7          return result;
8      }
9
10     public Result Add_block_to_chain_task(Result result) {
11         result.setSuccess();
12         return result;
```

```
    }

    public Result Notify_the_network_task(Result result) {
        result.setError("notify error");
        return result;
    }
}
```

Listing 5.3: Task class for the Orderer System

The activity diagram of the resulting MS4 project is present in Figure 5.11.

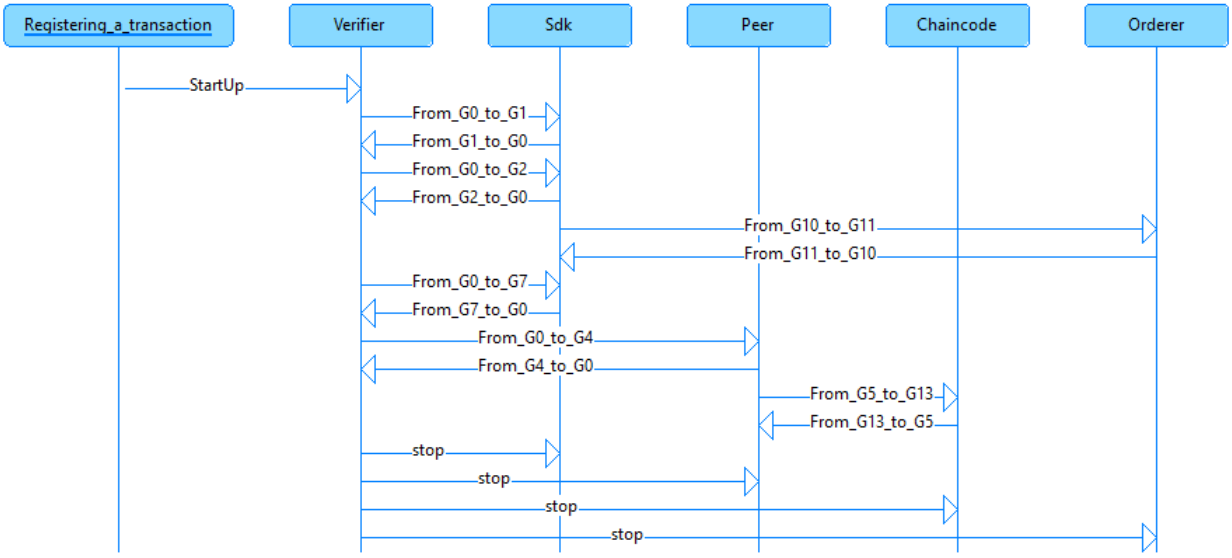

Figure 5.11: Activity diagram of the mission "Registering a Transaction"

In the following chapter it is presented the results of this work, such as the tests executed and the simulation output for the given cases.

Chapter 6

Results

In this chapter it is presented the results of the proposal, what kind of tests were applied, and what they assure about the application. Starting from the unit tests, where the tested cases are shown, to the simulation validation, where it is explained how to test the resulting project from the Section 5.1 in the MS4 ME and the simulation results.

6.1 Unit tests

Some simple unit tests were implemented in order to validate the input model and its conversion into a tree, including cases where it may fail or pass.

The cases contemplated at this stage were:

1. A model should not have more than one root to be loaded;
2. A loaded model must be converted into a tree without any errors;
3. Traversing the tree must be possible (which means it is valid); and
4. All the goal nodes under the tree must start with an identifier "Gn", where n is a natural number.

For the tests stated above, the tested models are the ones present in Appendices VII (an invalid model) and I (a valid model).

These basic checks prevent the user from feeding invalid models into the software. The checks may be expanded in future versions.

The result of the units tests is shown in the Figure 6.1.

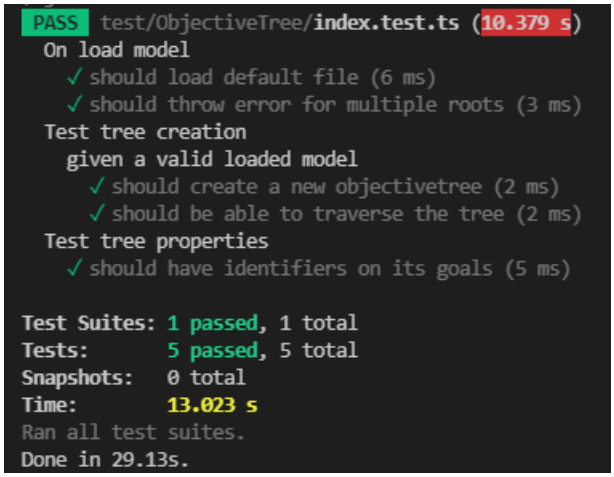

Figure 6.1: Unit tests result

6.2 Simulation validation

The model used as basis for the simulation is the example shown in Figure 3.6, its exported JSON file which was the input for the GM2MS4 tool is available in the Appendix I.

6.2.1 A blockchain mission

The proposed blockchain and its capabilities was presented before in Section 2.4. With the purpose of modeling an SoS mission, the model shown in Figure 3.6 represents the chosen mission for a blockchain network, in our example: How the network registers a new transaction into the history chain.

To register a transaction, a coordination of the constituent systems is necessary and it is performed by the application that is communicating with the blockchain (blockchain here means the set formed by the peers and orderers servers). This hypothetical application is named "SDK" (this is the value of the "component" property) in the model and it is represented by the green color ■.

The first goal to be achieved is to find the target peers for the transaction, that means, deciding which peers on the network will be responsible for calculating the transaction result for the customer request. The SDK can achieve this goal by any of the following means:

1. Connect to the discovery channel shared among the peers and the SDK to find peer candidates for the contract execution; or
2. Broadcast a search message to the blockchain network and choose the peers that responds to the broadcast message. The message can be sent through UDP or TCP; or
3. Try to reach the peers defined in a static configuration file.

This mission of the SDK is represented by the goal "G1: Find the target peers" at the example model.

If G1 is executed successfully then the next goal execution is started, the goal "G2: Send transaction request to the peers" takes the result of the G1 achievement and sends to those selected peers the transaction request to be calculated by the contract code of the blockchain network (also called smart-contracts or chaincode). The task of sending the request can be achieved by sending the request through gRPC[18] dialings or through REST requests.

For the next step "G4: Calculate transaction result" it is important to state that all participant peers of the blockchain network has an auxiliary system attached to it, this system is widely known as "smart-contract", in this work the name "chaincode" will be used. In the model, the goals of the peers can be identified by the yellow color , as for the goals of the chaincode, the blue color the color is used .

The goal G4 is decomposed into two goals:

1. "G5: invoke chaincode": requests to the chaincode system to calculate the transaction result, for example, the transference of a value from a user to another, changing the proprietary of the data; and
2. "G6: Return contract execution result": notifies the applications which are waiting for the transaction result about the contract execution return. This can be done by either broadcasting the result data to the network (in this case the application must be listening for the result) or by unicasting the message back to the application that have requested the transaction. The first is preferable since these are decentralized systems, but in the case of failure the second method can be applied to prevent the discarding of the data.

Note that both G5 and G6 should be executed successfully in order to achieve G4 accomplishment as they’re linked by an AND relation.

G5 also depends on the "G13: Execute business logic" goal to be achieved, which means that if the smart-contract execution fails, then the whole branch under G4 also fails, as G4 is related to the root node by an AND link, the failure of either G5 or G6 results in the simulation failure.

Going further on the model simulation, the next goal that needs to be achieved is "G7: Receive result pool", which is the state path of the simulation for the SDK system activated by the return of the chaincode execution.

As many peers were target to execute the transaction, the result arrives as a group of results. In a blockchain network, the transaction output must be deterministic, in other words: All the selected peers must return the same result for a given request, otherwise the peers replying with results diverging from the majority are considered malicious by the rest of the network. This verification process must be done in the task T11. In future works these verification tasks may be converted into model constraints in order to minimize the model complexity.

After verifying the results, if the pool validation passes, then the transaction result is passed down to the orderer system, which is responsible for validating the signatures present in the block. The consensus is the rule applied by the orderer on a incoming transaction. To be appended, it must be signed as the rule states,

the simplest rule is considered in this example: the majority of the peers must sign the transaction for it to reach the consensus.

The goal "G11: Validate block signatures" represents this process done by the orderer system (in the model, the goals of the orderer are colored in red ■). Once finished the appending result is sent back to the SDK that finishes the mission validation with Success if all the goals G0, G2, G4 and G7 were successfully achieved.

6.2.2 Simulation Results

To validate if the exported MS4 model was correctly analyzed, the mission execution, and the returned result from the tasks were forced to either Success or Fail states. The simulation must output a message containing the result of the running, informing if it was successfully executed for a given set of passing and failing tasks. **setSuccess** or **setError** for each task method present at the task classes of each constituent system. In the Table 6.1, the columns 1-13 represent the state forced onto the task. The expected simulation result and the executed result can be compared in the last two columns of Table.

It is noticeable that the tasks that compose OR relations affect less the accomplishment of the mission, that is, many can fail under that node given that the success of at least one of the tasks achieves the goal. While the tasks that are needed for the process to continue (the ones that are linked to its parent by an AND link) affect much more the mission accomplishment, in the sense that they are sensitive tasks on the mission evaluation. In other words, the failure of any of these tasks is sufficient to the mission failure as well.

The Table 6.1 shows the tested combinations of

Task Name	T0	T2	T3	T4	T6	T7	T8	T9	T10	T11	T12	T13	T14	Mission simulation expected result	Simulation Result
Task result	success	success	success	success	success	success	success	success	success	success	success	success	success	success	success
	success	fail	success	success	success	success	success	success	success	success	success	success	success	success	success
	success	fail	fail	success	success	success	success	success	success	success	success	success	success	success	success
	fail	fail	fail	fail	success	success	success	success	success	success	success	success	success	fail	fail
	success	fail	fail	fail	fail	success	success	success	success	success	success	success	success	success	success
	success	fail	fail	fail	fail	fail	success	success	success	success	success	success	success	fail	fail
	success	success	success	success	success	success	fail	success	success	success	success	success	success	fail	fail
	success	success	success	success	success	success	success	fail	success	success	success	success	success	success	success
	success	success	success	success	success	success	success	success	fail	success	success	success	success	success	success
	success	success	success	success	success	success	success	fail	fail	success	success	success	success	fail	fail
	success	success	success	success	success	success	success	success	success	fail	success	success	success	fail	fail
	success	success	success	success	success	success	success	success	success	success	fail	success	success	fail	fail
	success	success	success	success	success	success	success	success	success	success	success	fail	success	fail	fail
	success	success	success	success	success	success	success	success	success	success	success	success	fail	fail	fail

Table 6.1: Table result obtained by forcing tasks into either Failure or Success state

The initial state for the simulation is shown in the Figure 6.2. All the systems start from the same initial state "waitForInput".

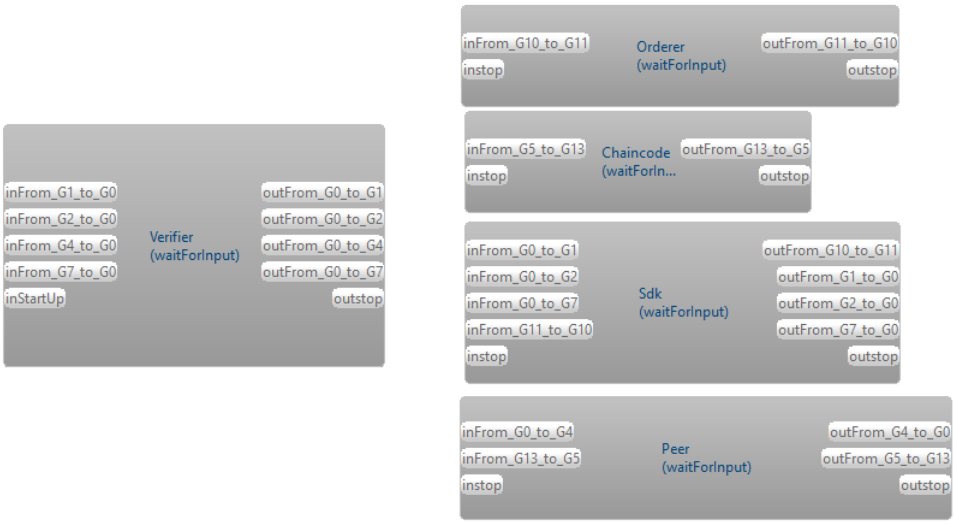

Figure 6.2: Simulation initial state

In the Figure 6.3, a view of the simulation running is presented.

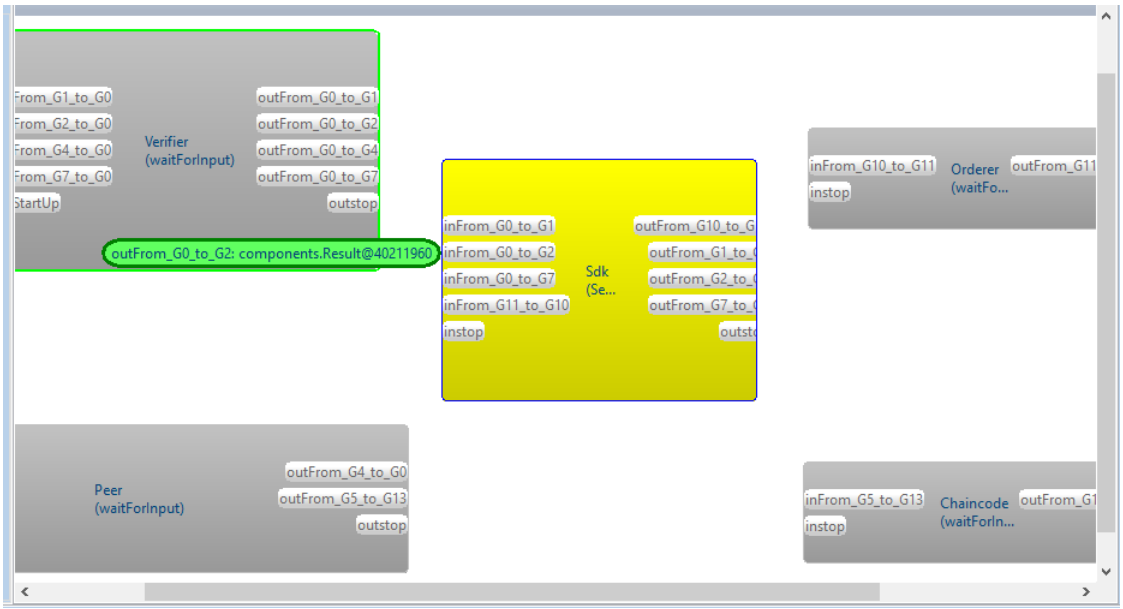

Figure 6.3: Verifier System sends "From_G0_to_G2" to the SDK system

Once finished, the simulation ends and all the constituent systems now passivate forever at the "stop" state. If needed, the simulation may be restarted by hitting the "Restart" button in the toolbar present at the bottle of the simulation.

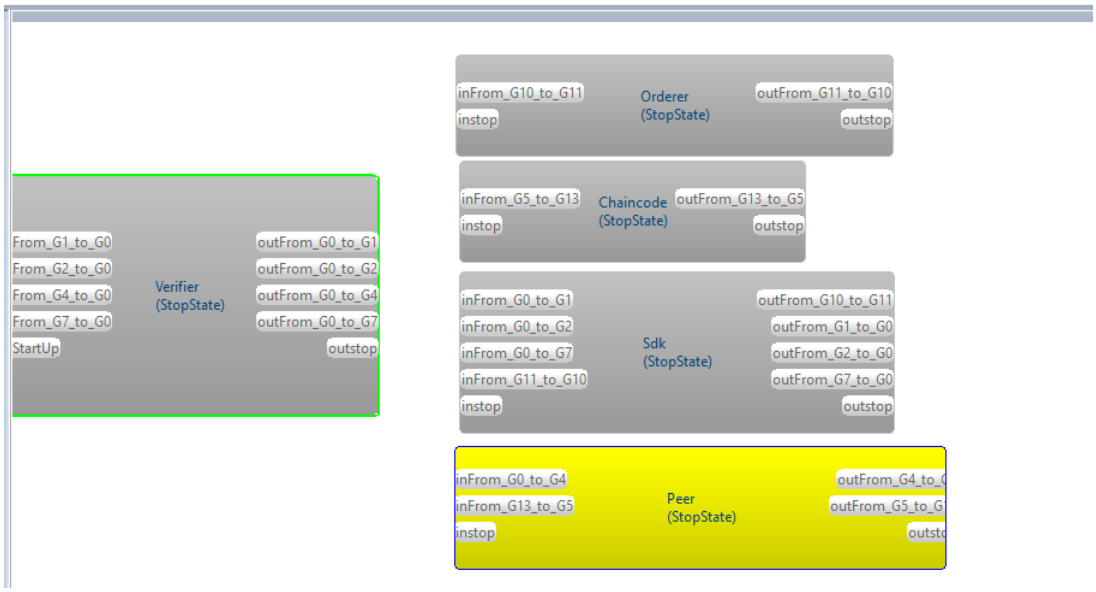

Figure 6.4: Simulation final state

The result shown in Table 6.1 corresponds to the expected behavior for the simulation, that is, the AND/OR relations of the model are respected, and they control the result of the simulation properly.

In Figure 6.5, the dialog windows for simulation Success and Failure results are shown. The windows appear after the simulation is finished.

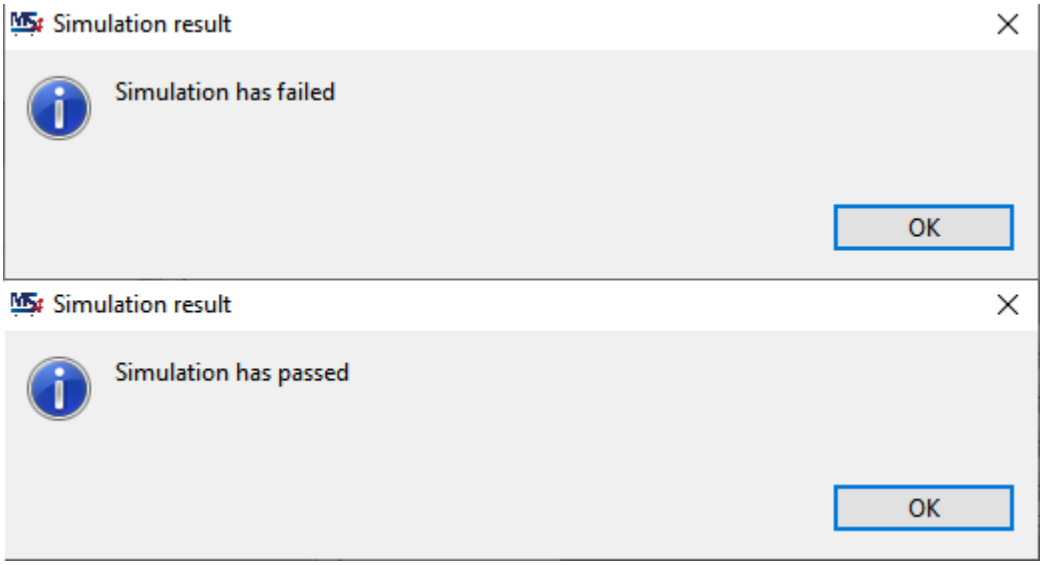

Figure 6.5: Simulation result dialog

The simulation result shown in Table 6.1 corresponds to the expected behavior designed in the goal model, which means that our goal was reached: From the proposed model it is possible to simulate an SoS from its mission perspective, implementing the tasks with Java code that can be used to interface with the real world. To simplify, the tests forced the tasks into a failure/success state - i.e. an

SoS designer may set the states by checking the response of an API or based on a serial communication to an IoT device or any other desired interface - and the result must match the expected one for that set of failures and successes.

6.3 Challenges

By far the hardest task in this work development was not the implementation of the converter itself, but:

1. Gathering and sorting out all the necessary information that would make it possible to create an MS4 project from the model.
2. Understanding the MS4 environment and gathering the necessary expertise to metaprogram its model. As the platform source is closed, the only resource for studying it is the software manuals and the modeling book recommended by the MS4 team [10].

As many files are outputted and linked together in the conversion process, one may think that verifying all the produced files present in the project (DNL, Java, and SES files) would be a tough task, but the functional programming style suggests to the developer to view their data as code: That is, the output is a sum of independent calculations over the properties of the elements, hence the need of all the nodes to have an identifier, task sequence, component name, and so on. These are common properties that occur in all nodes and which parametrize the calculations in the program. Combining this abstraction over the model data with the MS4 IDE features and compiler has made the work of projecting a compatible model for the MS4 platform easier.

Testing the simulation results was a time-demanding task, as there are thousands of failure and success combinations possible for the set of tasks present in the model, it is impossible for a human to test all of them and compile them into a result table. It may be interesting to implement scenarios generators that by providing the set of task results for a given context would facilitate the simulation verification process, and automatize it.

Chapter 7

Conclusion

This work proposes a model for missions of an SoS and provides a tool that converts the mission model to an MS4 ME project, which allows the simulation of the desired SoS, starting from its mission perspective. The research showed that a tool capable of chaining the mission modeling to the SoS simulation was inexistent, thus the implementation was challenging, not only because of the high level of abstraction and concepts mapping from the goal model to the MS4 syntax but also because a convertible mission model is missing in the literature.

The main benefit that the model and the GM2MS4 tool brings is to allow an SoS designer to describe it from the goals that must be achieved rather than describing the systems from its architectural perspective, as often in the SoS context the designer knows only the behaviors of the constituent systems and not its implementation. The MS4 project structure that the tool outputs makes it easier to interface the simulation with the real world while checking for the expected behavior printed in the model, this is a powerful feature that proves the model and that the designed goals are achievable.

In Table 7.1, the goals aimed at in Chapter 1, and the achieved results are compared.

Goal	Contributions	Results
Provide a goal model that has the necessary elements to specify a mission and the constituent systems relations present at this mission accomplishment.	A goal-oriented mission model capable of describing the expected behaviors for an SoS	The goal-oriented model proposed not only describes the mission from its expected behaviors but also respects many of the aspects proposed in [14] such as dependability between constituent systems and goal-achievement variance.

Provide a software solution that is capable of transforming this model into an MS4 simulation, expressing the expected behaviors present at the mission model in the view of each constituent system and how they are interconnected.	Tool implementation and testing	The implemented tool (GM2MS4) is capable of converting the SoS mission model into an SoS simulation under the MS4 systems
Verify these behaviors by adding the tasks present at the mission model as implementable code on the MS4 outputted project, allowing the design to interface the simulation with existent systems or self-implemented mocks, and to validate the mission accomplishment.	MS4 Java classes ready-to-code provided along with the simulation project	The leaf tasks present at the model have their correspondent methods in the tasks class for that component; The result returned by these tasks control the result of the simulation, providing information for the verifier node to evaluate the result of mission execution.
Test the proposed model and toolchain by implementing behaviors in the code of the tasks and verifying if the simulation result corresponds to the expected execution expressed in the model.	Testing Table 6.1	By evaluating the many combinations of tasks failure and success it is possible to prove the supplied model.

Table 7.1: Aimed goals and results

The software was implemented using Typescript, which is a very modern and flexible language. Combined with functional programming style it was possible to develop such a huge solution in a short time. The tool is easy to maintain due to its explicit types and transparent reference, which makes the code more readable. It is considered that the aim was achieved, even with all the challenges found along the way.

The limitations associated with the tool are the missing concepts for a mission modeling proposed by the mKaos [14], like constraints, contribution, and task parameters. Those must be analyzed and fit into the proposed model: how they can be expressed on the mission model built on the piStar tool. Once those elements are expressed in the model, the tool must be extended to translate them to the MS4 syntax.

For future works, it is suggested that these missing elements from the mission definition are targeted in order to provide robust SoS modeling. Although several checks are done in the software flow, it lacks unit testing to make the model rules more explicit for the developer working on it. Also, it is suggested to create documentation for the tool from the knowledge present in this manuscript.

It would be interesting to enable concurrent processes for OR junctions since currently the goals are executed sequentially following the order expressed by the tree nodes. As the OR junctions are independent processes, they may be executed at the same time without waiting for the preceding task to finish.

Finally, the implementation of a global static state is recommended, this would allow systems to perform checks on the SoS context to decide which strategy is the best to accomplish the goals.

References

[1] Russell L. Ackoff. Towards a System of Systems Concepts. *http://dx.doi.org/10.1287/mnsc.17.11.661*, 17(11):661–671, 7 1971. doi: 10.1287/MNSC.17.11.661. URL https://pubsonline.informs.org/doi/abs/10.1287/mnsc.17.11.661. 4, 6

[2] Suhair Alhajhassan, Mohammad Odeh, and Stewart Green. Aligning systems of systems engineering with goal-oriented approaches using the i framework. *ISSE 2016 - 2016 International Symposium on Systems Engineering - Proceedings Papers*, 11 2016. doi: 10.1109/SYSENG.2016.7753125. 10

[3] John Boardman and Brian Sauser. System of Systems - The meaning of of. In *Proceedings 2006 IEEE/SMC International Conference on System of Systems Engineering*, volume 2006, 2006. doi: 10.1109/sysose.2006.1652284. 5

[4] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems 2004 8:3*, 8(3):203–236, 5 2004. ISSN 1573-7454. doi: 10.1023/B:AGNT.0000018806.20944.EF. URL https://link.springer.com/article/10.1023/B:AGNT.0000018806.20944.ef. 9

[5] Steve Cook, Conrad Bock, Pete Rivett, Tom Rutt, Ed Seidewitz, Bran Selic, and Doug Tolbert. Unified Modeling Language (UML) Version 2.5.1. Technical report, Object Management Group (OMG), 12 2017. URL https://www.omg.org/spec/UML/2.5.1. 11

[6] Fabiano Dalpiaz, Xavier Franch, and Jennifer Horkoff. iStar 2.0 Language Guide. 10 2016. 9

[7] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde. GRAIL/KAOS. pages 612–613, 1997. doi: 10.1145/253228.253499. 7

[8] Mark W. Maier. Architecting Principles for Systems-of-Systems. *INCOSE International Symposium*, 6(1):565–573, 7 1996. doi: 10.1002/J.2334-5837.1996.TB02054.X. 1, 4, 5

[9] MS4 Systems Inc. Welcome to MS4 Systems. URL http://www.ms4systems.com/pages/main.php. 9, 10

[10] Bernard P. Zeigler and Hessam S. Sarjoughian. Guide to Modeling and Simulation of Systems of Systems. 2017. doi: 10.1007/978-3-319-64134-8. URL http://link.springer.com/10.1007/978-3-319-64134-8. 13, 44

[11] Joao Pimentel and Jaelson Castro. PiStar tool - A pluggable online tool for goal modeling. *Proceedings - 2018 IEEE 26th International Requirements Engineering Conference, RE 2018*, pages 498–499, 10 2018. doi: 10.1109/RE.2018.00071. 9, 60

[12] Leandro Santos Bergmann and Orientador Profa Dra Genaína Nunes Rodrigues Brasília. piStar-GODA: Integração entre os projetos piStar e GODA. Technical report, 2018. 19

[13] Leandro Santos Bergmann and Genaína Nunes Rodrigues. piStar-GODA: Integração entre os projetos piStar e GODA. Technical report, Universidade de Brasília, Brasília, 3 2018. URL https://bdm.unb.br/handle/10483/20428. 9, 25

[14] Eduardo Silva, Everton Cavalcante, Thais Batista, Flavio Oquendo, Flavia C. Delicato, and Paulo F. Pires. On the characterization of missions of systems-of-systems. In *ACM International Conference Proceeding Series*. Association for Computing Machinery, 2014. ISBN 9781450327787. doi: 10.1145/2642803.2642829. iii, iv, vii, 2, 7, 8, 45, 46

[15] Eduardo Silva, Thais Batista, and Everton Cavalcante. A Mission-Oriented Tool for System-of-Systems Modeling. In *Proceedings - 3rd International Workshop on Software Engineering for Systems-of-Systems, SESoS 2015*, 2015. doi: 10.1109/SESoS.2015.13. vii, 7, 9

[16] Ben van Lier. Blockchain Technology: The Autonomy and Self-Organisation of Cyber-Physical Systems. *Business Transformation through Blockchain*, pages 145–167, 2019. doi: 10.1007/978-3-319-98911-2{_}5. URL https://link.springer.com/chapter/10.1007/978-3-319-98911-2_5. 15

[17] Manoel Vieira. GM2MS4 (Goal Model to MS4), 11 2021. URL https://github.com/vieirin/GM2MS4. 30

[18] WangXingwei, ZhaoHong, and ZhuJiakeng. GRPC. *ACM SIGOPS Operating Systems Review*, 27(3):75–86, 7 1993. doi: 10.1145/155870.155881. URL https://dl.acm.org/doi/abs/10.1145/155870.155881. 40

[19] Bernard P. Zeigler, Hae Sang Song, Tag Gon Kim, and Herbert Praehofer. DEVS framework for modelling, simulation, analysis, and design of hybrid systems. pages 529–551, 10 1994. doi: 10.1007/3-540-60472-3{_}27. URL https://link.springer.com/chapter/10.1007/3-540-60472-3_27. 10

Appendix I

GM2MS4 Input file

```
{
  "actors": [
    {
      "id": "309c8d4d-f7a1-4ca3-83a8-a93e9240716b",
      "text": "Registering a transaction",
      "type": "istar.Actor",
      "x": 65,
      "y": 261,
      "nodes": [
        {
          "id": "f0406552-00af-4442-b03d-a78e27457929",
          "text": "G0: Register a transaction [G1;G2;G4;G7]",
          "type": "istar.Goal",
          "x": 653,
          "y": 276,
          "customProperties": {
            "Description": "",
            "selected": true,
            "component": "verifier"
          }
        },
        {
          "id": "2bb58c75-9a35-4b6f-8e8c-85f4cb604c84",
          "text": "G4: Calculate transaction result [G5;G6]",
          "type": "istar.Goal",
          "x": 676,
          "y": 398,
          "customProperties": {
            "Description": "",
            "component": "peer"
          }
        },
        {
```

```
          "id": "51e1a122-2abe-4153-a42e-3e0c7a3bbd8e",
          "text": "G5: Invoke chaincode",
          "type": "istar.Goal",
          "x": 698,
          "y": 484,
          "customProperties": {
            "Description": "",
            "component": "peer"
          }
        },
        {
          "id": "52781a22-2ed1-4281-9c73-e6a9d314e240",
          "text": "G6: Return contract execution result [T9;T10]",
          "type": "istar.Goal",
          "x": 854,
          "y": 508,
          "customProperties": {
            "Description": "component",
            "peer": "",
            "component": "peer"
          }
        },
        {
          "id": "7c79330d-f564-4134-b810-47370afd7308",
          "text": "G8: Verify result pool [T11;G10]",
          "type": "istar.Goal",
          "x": 1019,
          "y": 339,
          "customProperties": {
            "Description": "",
            "component": "api"
          }
        },
        {
          "id": "30fdd731-24c7-424d-b77c-6b9683a76bca",
          "text": "G10: Send signed transaction to Orderer",
          "type": "istar.Goal",
          "x": 1128,
          "y": 420,
          "customProperties": {
            "Description": "",
            "component": "api"
          }
        },
        {
          "id": "fc1c1d20-d10d-48d0-a717-59094a59d944",
```

```
          "text": "G11: Validate block signatures [T12;G12]",
          "type": "istar.Goal",
          "x": 1229,
          "y": 528,
          "customProperties": {
            "Description": "",
            "component": "orderer"
          }
        },
        {
          "id": "0973aeef-286f-47b0-a6cf-640e43d7671f",
          "text": "G12: Create block [T13;T14]",
          "type": "istar.Goal",
          "x": 1329,
          "y": 611,
          "customProperties": {
            "Description": "",
            "compoonent": "orderer",
            "component": "orderer"
          }
        },
        {
          "id": "c4146a67-384f-48f8-b48a-65fbdb31c7d0",
          "text": "T13: Add block to chain",
          "type": "istar.Task",
          "x": 1298,
          "y": 677,
          "customProperties": {
            "Description": "",
            "component": "orderer"
          }
        },
        {
          "id": "637bd788-65ab-4223-81cd-ceab64eef5fb",
          "text": "T14:  Notify the network",
          "type": "istar.Task",
          "x": 1398,
          "y": 673,
          "customProperties": {
            "Description": "",
            "component": "orderer"
          }
        },
        {
          "id": "3867e9d3-349c-4829-8c9b-61de16ac60c2",
          "text": "G13: Execute bussiness logic",
```

```
          "type": "istar.Goal",
          "x": 701,
          "y": 559,
          "customProperties": {
            "Description": "",
            "component": "chaincode"
          }
        },
        {
          "id": "ba3fd6ff-acbc-4a10-911a-55c4b655ed8c",
          "text": "T8: Execute requested contract function",
          "type": "istar.Task",
          "x": 667,
          "y": 741,
          "customProperties": {
            "Description": "",
            "component": "chaincode"
          }
        },
        {
          "id": "bdacda33-50aa-4850-9ce5-ad6a47368972",
          "text": "T9: Broadcast transaction result pool to channel",
          "type": "istar.Task",
          "x": 774,
          "y": 626,
          "customProperties": {
            "Description": "",
            "component": "peer"
          }
        },
        {
          "id": "0c59d725-3739-47ad-8ce9-d49b65004a52",
          "text": "G7: Receive result pool",
          "type": "istar.Goal",
          "x": 867,
          "y": 317,
          "customProperties": {
            "Description": "",
            "component": "api",
            "receives": "proposalResult"
          }
        },
        {
          "id": "7cdffcb0-d59f-4ee1-aaff-da02092465b0",
          "text": "T10: Unicast transaction result through grpc",
          "type": "istar.Task",
```

```
          "x": 935,
          "y": 688,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "859d8dee-5dc8-4b11-976b-47d3a4e76d25",
          "text": "G1: Find the target peers [T0;T1;T2]",
          "type": "istar.Goal",
          "x": 223,
          "y": 338,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "9b37915e-83b7-4055-8a86-ffaafdd5d377",
          "text": "T0: Search for peers on discovery channel",
          "type": "istar.Task",
          "x": 82,
          "y": 425,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "fecf3fa3-1a75-45d9-a6f4-05c0b55c4991",
          "text": "T1: Broadcast search message [T3;T4]",
          "type": "istar.Task",
          "x": 219,
          "y": 415,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "27c52174-a655-4fa4-967f-f0a919ed3425",
          "text": "T2: Extract from local config file",
          "type": "istar.Task",
          "x": 318,
          "y": 450,
          "customProperties": {
            "Description": ""
          }
        },
```

```
        {
          "id": "0bda1fdd-58bc-4a29-9177-a6c09c88f6f8",
          "text": "G2: Send trasaction request to peers",
          "type": "istar.Goal",
          "x": 470,
          "y": 417,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "17a290a1-59cb-4128-9b2c-312dce0bd36e",
          "text": "T3: Broadcast through TCP",
          "type": "istar.Task",
          "x": 162,
          "y": 538,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "f6a65702-e439-4abe-9600-99f83cafabdd",
          "text": "T4: Broadcast through UDP",
          "type": "istar.Task",
          "x": 277,
          "y": 540,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "7e46c2d1-1b9b-4b69-b97f-013c56a22d7e",
          "text": "T5: Send request through TCP [T6;T7]",
          "type": "istar.Task",
          "x": 477,
          "y": 536,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "3f2c7cf3-5898-45e3-bc25-47c5f456df9f",
          "text": "T6: Send request through gRPC",
          "type": "istar.Task",
          "x": 389,
          "y": 650,
```

```
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "92d7e548-b52d-448e-93fd-3f260694f605",
          "text": "T7: Send request through REST",
          "type": "istar.Task",
          "x": 562,
          "y": 663,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "439736ed-a61f-4cbf-bf4b-559506be953a",
          "text": "T11: Verify peer response",
          "type": "istar.Task",
          "x": 971,
          "y": 431,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "44a63bbf-dffe-48a0-af55-633460d4dec5",
          "text": "T12: Check signatures",
          "type": "istar.Task",
          "x": 1188,
          "y": 626,
          "customProperties": {
            "Description": ""
          }
        }
      ]
    }
  ],
  "links": [
    {
      "id": "fa476696-fe1f-47a3-ab4d-e784b2ff8986",
      "type": "istar.AndRefinementLink",
      "source": "2bb58c75-9a35-4b6f-8e8c-85f4cb604c84",
      "target": "f0406552-00af-4442-b03d-a78e27457929"
    },
    {
      "id": "4cea1497-9abf-496e-a8f0-094dbecc3588",
```

```
      "type": "istar.AndRefinementLink",
      "source": "51e1a122-2abe-4153-a42e-3e0c7a3bbd8e",
      "target": "2bb58c75-9a35-4b6f-8e8c-85f4cb604c84"
    },
    {
      "id": "bac1272c-1ce2-4a26-bdb9-38d641c9e35e",
      "type": "istar.AndRefinementLink",
      "source": "52781a22-2ed1-4281-9c73-e6a9d314e240",
      "target": "2bb58c75-9a35-4b6f-8e8c-85f4cb604c84"
    },
    {
      "id": "61e14467-0a96-40cb-8bbe-11084b7656ae",
      "type": "istar.AndRefinementLink",
      "source": "fc1c1d20-d10d-48d0-a717-59094a59d944",
      "target": "30fdd731-24c7-424d-b77c-6b9683a76bca"
    },
    {
      "id": "d1df420d-447a-4620-9705-82ce8ff0f01c",
      "type": "istar.AndRefinementLink",
      "source": "3867e9d3-349c-4829-8c9b-61de16ac60c2",
      "target": "51e1a122-2abe-4153-a42e-3e0c7a3bbd8e"
    },
    {
      "id": "876e5e77-4141-40f9-b5f4-3c97f5a87c67",
      "type": "istar.AndRefinementLink",
      "source": "ba3fd6ff-acbc-4a10-911a-55c4b655ed8c",
      "target": "3867e9d3-349c-4829-8c9b-61de16ac60c2"
    },
    {
      "id": "41c0afb7-9e89-4b12-8cda-13fa24a4a0d1",
      "type": "istar.AndRefinementLink",
      "source": "0c59d725-3739-47ad-8ce9-d49b65004a52",
      "target": "f0406552-00af-4442-b03d-a78e27457929"
    },
    {
      "id": "4466f6ec-58f8-4909-a90c-0da8a0fe35c8",
      "type": "istar.AndRefinementLink",
      "source": "7c79330d-f564-4134-b810-47370afd7308",
      "target": "0c59d725-3739-47ad-8ce9-d49b65004a52"
    },
    {
      "id": "d61e1fe5-603b-43a1-9d6f-b265f030e49c",
      "type": "istar.AndRefinementLink",
      "source": "c4146a67-384f-48f8-b48a-65fbdb31c7d0",
      "target": "0973aeef-286f-47b0-a6cf-640e43d7671f"
    },
```

```
    {
      "id": "951d4f12-a130-4a70-9e28-2adaafa67906",
      "type": "istar.AndRefinementLink",
      "source": "637bd788-65ab-4223-81cd-ceab64eef5fb",
      "target": "0973aeef-286f-47b0-a6cf-640e43d7671f"
    },
    {
      "id": "379b5b86-164e-4e8a-bf87-78fdf4dd240c",
      "type": "istar.OrRefinementLink",
      "source": "bdacda33-50aa-4850-9ce5-ad6a47368972",
      "target": "52781a22-2ed1-4281-9c73-e6a9d314e240"
    },
    {
      "id": "fec075a9-7334-4053-b622-d30efe954698",
      "type": "istar.OrRefinementLink",
      "source": "7cdffcb0-d59f-4ee1-aaff-da02092465b0",
      "target": "52781a22-2ed1-4281-9c73-e6a9d314e240"
    },
    {
      "id": "db76bbf4-81d5-4c53-9909-45c6eb8fde3e",
      "type": "istar.AndRefinementLink",
      "source": "859d8dee-5dc8-4b11-976b-47d3a4e76d25",
      "target": "f0406552-00af-4442-b03d-a78e27457929"
    },
    {
      "id": "6fcf5fbe-d35b-4ddc-8ce2-c83719edd039",
      "type": "istar.OrRefinementLink",
      "source": "9b37915e-83b7-4055-8a86-ffaafdd5d377",
      "target": "859d8dee-5dc8-4b11-976b-47d3a4e76d25"
    },
    {
      "id": "ea4e222f-c86a-4225-a8f6-afd28ba3e0f6",
      "type": "istar.OrRefinementLink",
      "source": "fecf3fa3-1a75-45d9-a6f4-05c0b55c4991",
      "target": "859d8dee-5dc8-4b11-976b-47d3a4e76d25"
    },
    {
      "id": "dd20196e-e2cf-417f-bcc0-d28fb7068c73",
      "type": "istar.OrRefinementLink",
      "source": "27c52174-a655-4fa4-967f-f0a919ed3425",
      "target": "859d8dee-5dc8-4b11-976b-47d3a4e76d25"
    },
    {
      "id": "f8607312-ac78-44c4-8b46-124fa4585a07",
      "type": "istar.OrRefinementLink",
      "source": "17a290a1-59cb-4128-9b2c-312dce0bd36e",
```

```
      "target": "fecf3fa3-1a75-45d9-a6f4-05c0b55c4991"
    },
    {
      "id": "23e34400-6652-4cf9-a697-7a31a21d71e9",
      "type": "istar.OrRefinementLink",
      "source": "f6a65702-e439-4abe-9600-99f83cafabdd",
      "target": "fecf3fa3-1a75-45d9-a6f4-05c0b55c4991"
    },
    {
      "id": "898e01ec-8d4f-44e3-bd83-05786b374530",
      "type": "istar.AndRefinementLink",
      "source": "0bda1fdd-58bc-4a29-9177-a6c09c88f6f8",
      "target": "f0406552-00af-4442-b03d-a78e27457929"
    },
    {
      "id": "cfe65907-64fd-4e1f-a68c-e16ad564128a",
      "type": "istar.OrRefinementLink",
      "source": "3f2c7cf3-5898-45e3-bc25-47c5f456df9f",
      "target": "7e46c2d1-1b9b-4b69-b97f-013c56a22d7e"
    },
    {
      "id": "e97bbe43-6d00-4eb9-b788-d79d3476b674",
      "type": "istar.OrRefinementLink",
      "source": "92d7e548-b52d-448e-93fd-3f260694f605",
      "target": "7e46c2d1-1b9b-4b69-b97f-013c56a22d7e"
    },
    {
      "id": "07a33966-7d8d-427e-be6a-8f250b9e5c71",
      "type": "istar.AndRefinementLink",
      "source": "7e46c2d1-1b9b-4b69-b97f-013c56a22d7e",
      "target": "0bda1fdd-58bc-4a29-9177-a6c09c88f6f8"
    },
    {
      "id": "1fe3926f-a1b4-4a33-b487-44341f26e40a",
      "type": "istar.AndRefinementLink",
      "source": "439736ed-a61f-4cbf-bf4b-559506be953a",
      "target": "7c79330d-f564-4134-b810-47370afd7308"
    },
    {
      "id": "e6e3bf53-a894-4ff2-9220-c980625bc057",
      "type": "istar.AndRefinementLink",
      "source": "30fdd731-24c7-424d-b77c-6b9683a76bca",
      "target": "7c79330d-f564-4134-b810-47370afd7308"
    },
    {
      "id": "5dd177dc-5aa1-492f-98cb-b967f1f17939",
```

```
      "type": "istar.AndRefinementLink",
      "source": "44a63bbf-dffe-48a0-af55-633460d4dec5",
      "target": "fc1c1d20-d10d-48d0-a717-59094a59d944"
    },
    {
      "id": "7b02d3cc-e8e6-4724-881a-b7191b93ea5c",
      "type": "istar.AndRefinementLink",
      "source": "0973aeef-286f-47b0-a6cf-640e43d7671f",
      "target": "fc1c1d20-d10d-48d0-a717-59094a59d944"
    }
  ],
  [...]
  }
}
```

Listing I.1: Model converter input file

Some unused fields are omitted, please refer to piStar work[11] for the complete file.

Appendix II

Peer DNL file

```
use peer_transition with type  components.PeerTransitionsClass and default "new components.PeerTransitionsClass()"!
use result with type  components.Result and default "new components.Result(\"peer\")"!

To start passivate in waitForInput!
Passivate in StopState!


when in waitForInput and receive From_G0_to_G4 go to Calculate_transaction_result!
external event for waitForInput with From_G0_to_G4
<%
    Result incomingResult = result.update(messageList.get(0).getData());
    result.reset(incomingResult);

%>!

when in waitForInput and receive From_G13_to_G5 go to Execute_bussiness_logic_continue!
external event for waitForInput with From_G13_to_G5
<%
    result = result.update(messageList.get(0).getData());

%>!

when in waitForInput and receive stop go to StopState!

generates output on From_G5_to_G13 with type Result!
generates output on From_G4_to_G0 with type Result!
generates output on stop !

```

```
accepts input on From_G13_to_G5 with type Result !
accepts input on From_G0_to_G4 with type Result !


hold in Calculate_transaction_result for time 5!
from Calculate_transaction_result go to Invoke_chaincode!
internal event for Calculate_transaction_result
<%
    peer_transition.calculate_transaction_result_runner(result);
%>!


hold in Invoke_chaincode for time 5!
from Invoke_chaincode go to waitForInput!
internal event for Invoke_chaincode
<%
    peer_transition.invoke_chaincode_runner(result);
%>!

after Invoke_chaincode output From_G5_to_G13!
output event for Invoke_chaincode
<%
    output.add(outFrom_G5_to_G13, result);
%>!




hold in Execute_bussiness_logic_continue for time 5!
from Execute_bussiness_logic_continue go to
   Return_contract_execution_result!
internal event for Execute_bussiness_logic_continue
<%
    peer_transition.execute_bussiness_logic_continue_runner(result);
%>!


hold in Return_contract_execution_result for time 5!
from Return_contract_execution_result go to waitForInput!
internal event for Return_contract_execution_result
<%
    peer_transition.return_contract_execution_result_runner(result);
%>!

after Return_contract_execution_result output From_G4_to_G0!
output event for Return_contract_execution_result
<%
```

```
74      output.add(outFrom_G4_to_G0, result);
75  %>!
```

Listing II.1: DNL file for peer component in model 3.6

Appendix III

Peer Transitions Class

```
1  package components;
2
3  public class PeerTransitionsClass {
4
5
6      interface TaskRunner {
7          Result run(Result res);
8      }
9
10     private peerTaskClass PeerRunner = new peerTaskClass();
11
12     private Result tasksRunner (TaskRunner[] tasks, String relation,
           String parentRelation,Result result){
13         Result lastRes = result;
14         for (TaskRunner run : tasks) {
15             Result res = run.run(lastRes);
16
17             res = verifyContinuation(res, relation, parentRelation ==
                   "and");
18
19             lastRes.update(res);
20             if (res.locked()) {
21                 break;
22             }
23
24         }
25         return lastRes;
26     }
27     private Result verifyContinuation(Result result, String relation,
           boolean canLock) {
28     if (((result.getError() == null && result.isSuccess())&& relation
           == "or") || ((result.getError() != null &&
           !result.isSuccess())&& relation == "and")) {
```

```
        if (canLock) {
            result.lock();
        }
    }
    // before continuing to next functions
    if (!result.locked()) {
        result.resetError();
    }
    return result;
}

    public Result calculate_transaction_result_runner(Result result) {


        if (result.locked()) {
            return result;
        }

        return result;

        //Goes to state: Invoke_chaincode
    }

    public Result invoke_chaincode_runner(Result result) {

       result = verifyContinuation(result, "and" , true);
        if (result.locked()) {
            return result;
        }

        return result;

        //Goes to state: Execute_bussiness_logic
    }

    public Result execute_bussiness_logic_continue_runner(Result
       result) {

       result = verifyContinuation(result, "and" , true);
        if (result.locked()) {
            return result;
        }

        return result;

        //Goes to state: output_state
```

```
74      }
75
76
77      public Result return_contract_execution_result_runner(Result
            result) {
78
79          result = verifyContinuation(result, "and" , true);
80           if (result.locked()) {
81               return result;
82           }
83
84           TaskRunner[] runners = new TaskRunner[] {
85              new TaskRunner() {
86               public Result run(Result res) {
87                   return PeerRunner.Broadcast_transaction
88                       _result_pool_to_channel_task(res);
89                   }
90              },
91              new TaskRunner() {
92               public Result run(Result res) {
93                   return PeerRunner.Unicast_transaction
94                       _result_through_grpc_task(res);
95                   }
96              }
97           };
98
99           return tasksRunner(runners, "or", "and", result);
100
101          //Goes to state: output_state
102     }
103
104
105 }
```

Listing III.1: PeerTranstionsClass.java

Appendix IV

Peer task class

```
1  package components;
2
3  public class peerTaskClass {
4
5      public Result Broadcast_transaction_result_pool_to_channel_task
           (Result result) {
6              return result;
7      }
8
9      public Result Unicast_transaction_result_through_grpc_task
           (Result result) {
10             return result;
11     }
12 }
```

Listing IV.1: peerTaskClass.java

Appendix V

Result class

```
1  package components;
2
3  import java.io.Serializable;
4
5  public class Result implements Serializable {
6
7      private static final long serialVersionUID = 5018535970263352859L;
8
9      private ErrorSignal error = null;
10     private String component = "";
11     private boolean success = false;
12     private boolean isLocked = false;;
13     private String result = "";
14
15     public void setError(String error) {
16         System.out.println(error);
17         this.error = new ErrorSignal(error, this.component);
18         this.success = false;
19         this.result = error;
20     }
21
22     public ErrorSignal getError() {
23         return error;
24     }
25
26     public void setSuccess() {
27         if (!isLocked) {
28             error = null;
29             success = true;
30             System.out.println("done with success");
31         }
32     }
33
```

```
34      public void setSuccess(String result) {
35          if (!isLocked) {
36              success = true;
37              this.result = result;
38          }
39          setSuccess();
40      }
41
42      public void lock() {
43          this.isLocked = true;
44      }
45
46      public boolean locked() {
47          return this.isLocked;
48      }
49
50      public boolean isSuccess() {
51          return success;
52      }
53
54      public void resetError() {
55          this.error = null;
56
57      }
58
59      public void setResult(String result) {
60          if (!this.isLocked) {
61              this.result = result;
62          }
63      }
64
65      public String getResult() {
66          return result;
67      }
68
69      public void reset (Result res) {
70          this.error = null;
71          this.success = false;
72          this.isLocked = false;
73          this.result = res.result;
74      }
75
76      public Result update(Result res) {
77          if (!this.isLocked) {
78              this.error = res.error;
79              this.success = res.success;
```

```
80              this.result = res.result;
81          }
82          return this;
83      }
84
85      public Result (String  component) {
86          this.component = component;
87      }
88
89      public Result (String  component, String initialState) {
90          result = initialState;
91          this.component = component;
92      }
93
94
95      public void print() {
96          System.out.print("Succes: ");
97          System.out.print(success);
98          System.out.print(" Locked: ");
99          System.out.print(isLocked);
100         System.out.print(" Error: ");
101
102         System.out.print(error != null ? error.error : "null" );
103     }
104 }
```

Listing V.1: Result.java

Appendix VI

Error Class

```java
package components;

public class ErrorSignal {
    public String error;
    public String origin;
    public ErrorSignal (String  error, String origin) {
        this.error = error;
        this.origin = origin;
    }
}
```

Listing VI.1: Error.java

Appendix VII

Invalid model

```
{
  "actors": [
    {
      "id": "309c8d4d-f7a1-4ca3-83a8-a93e9240716b",
      "text": "Registrando uma transa  o",
      "type": "istar.Actor",
      "x": 103,
      "y": 152,
      "customProperties": {
        "Description": ""
      },
      "nodes": [
        {
          "id": "f0406552-00af-4442-b03d-a78e27457929",
          "text": "Registrar uma transa  o",
          "type": "istar.Goal",
          "x": 416,
          "y": 170,
          "customProperties": {
            "Description": "",
            "selected": true
          }
        },
        {
          "id": "8de7a901-398e-4d37-a22d-3f954bb81601",
          "text": "Processar entrada na api",
          "type": "istar.Goal",
          "x": 268,
          "y": 233,
          "customProperties": {
            "Description": "",
            "selected": true
          }
```

```
      },
      {
        "id": "2bb58c75-9a35-4b6f-8e8c-85f4cb604c84",
        "text": "Calcular resultado da transa  o",
        "type": "istar.Goal",
        "x": 470,
        "y": 230,
        "customProperties": {
          "Description": ""
        }
      },
      {
        "id": "51e1a122-2abe-4153-a42e-3e0c7a3bbd8e",
        "text": "Invocar o chaincode",
        "type": "istar.Goal",
        "x": 501,
        "y": 322,
        "customProperties": {
          "Description": ""
        }
      },
      {
        "id": "52781a22-2ed1-4281-9c73-e6a9d314e240",
        "text": "Enviar resultado para api",
        "type": "istar.Goal",
        "x": 606,
        "y": 325,
        "customProperties": {
          "Description": ""
        }
      },
      {
        "id": "7c79330d-f564-4134-b810-47370afd7308",
        "text": "Verificar a pool de resultados",
        "type": "istar.Goal",
        "x": 771,
        "y": 236,
        "customProperties": {
          "Description": ""
        }
      },
      {
        "id": "0d5629f0-4d07-46e9-a452-cac4d322e234",
        "text": "Rejeitar a transa  o",
        "type": "istar.Goal",
        "x": 726,
```

```
          "y": 315,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "30fdd731-24c7-424d-b77c-6b9683a76bca",
          "text": "Enviar transa  o assinada para o orderer",
          "type": "istar.Goal",
          "x": 879,
          "y": 309,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "fc1c1d20-d10d-48d0-a717-59094a59d944",
          "text": "Validar assinaturas do bloco",
          "type": "istar.Goal",
          "x": 895,
          "y": 372,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "0973aeef-286f-47b0-a6cf-640e43d7671f",
          "text": "Rejeitar a transa  o",
          "type": "istar.Goal",
          "x": 813,
          "y": 448,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "a219eddf-f170-454f-b20b-7568228307e0",
          "text": "Criar bloco",
          "type": "istar.Task",
          "x": 974,
          "y": 454,
          "customProperties": {
            "Description": ""
          }
        },
        {
```

```
          "id": "c4146a67-384f-48f8-b48a-65fbdb31c7d0",
          "text": "Adicionar bloco    cadeia",
          "type": "istar.Task",
          "x": 923,
          "y": 518,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "637bd788-65ab-4223-81cd-ceab64eef5fb",
          "text": "Notificar a rede ",
          "type": "istar.Task",
          "x": 1046,
          "y": 520,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "0429a68a-713a-4918-96d9-f3fbff61569d",
          "text": "Abortar bad input",
          "type": "istar.Goal",
          "x": 107,
          "y": 296,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "f6f3d933-8b3d-4c61-893c-cca36d2972c8",
          "text": "Montar proposta de transa  o",
          "type": "istar.Goal",
          "x": 273,
          "y": 302,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "c4963052-bad5-46ef-b79e-031266546f11",
          "text": "Montar proposta de transa  o",
          "type": "istar.Task",
          "x": 213,
          "y": 370,
          "customProperties": {
```

```
            "Description": ""
          }
        },
        {
          "id": "9c1403b9-53f3-4036-8690-3af589e5c536",
          "text": "Abort (input)",
          "type": "istar.Task",
          "x": 103,
          "y": 374,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "3867e9d3-349c-4829-8c9b-61de16ac60c2",
          "text": "Executar l gica de neg cio",
          "type": "istar.Goal",
          "x": 503,
          "y": 392,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "ba3fd6ff-acbc-4a10-911a-55c4b655ed8c",
          "text": "Executar  fun   o solicitada",
          "type": "istar.Task",
          "x": 502,
          "y": 454,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "e3ea007b-e9df-4651-9260-8566a86c52d8",
          "text": "Enviar proposta para os Peers",
          "type": "istar.Goal",
          "x": 334,
          "y": 362,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "bbfc14b1-9da7-4112-82cf-3ec369887261",
          "text": "Enviar proposta para os peers-alvo",
```

```
          "type": "istar.Task",
          "x": 403,
          "y": 452,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "b96e97cd-d783-4f76-b006-b4c0ef93f4c9",
          "text": "Calcular peers-alvo",
          "type": "istar.Task",
          "x": 258,
          "y": 451,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "bdacda33-50aa-4850-9ce5-ad6a47368972",
          "text": "Enviar resultado para api",
          "type": "istar.Task",
          "x": 615,
          "y": 397,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "0c59d725-3739-47ad-8ce9-d49b65004a52",
          "text": "Receber pool de resultados",
          "type": "istar.Goal",
          "x": 663,
          "y": 218,
          "customProperties": {
            "Description": ""
          }
        },
        {
          "id": "72f6d345-7050-4382-8593-10719959349b",
          "text": "Enviar erro",
          "type": "istar.Task",
          "x": 724,
          "y": 395,
          "customProperties": {
            "Description": ""
          }
```

```
        }
      ]
    }
  ],
  "links": [
    {
      "id": "69723206-a23d-49da-960e-4a9c4d569598",
      "type": "istar.AndRefinementLink",
      "source": "8de7a901-398e-4d37-a22d-3f954bb81601",
      "target": "f0406552-00af-4442-b03d-a78e27457929"
    },
    {
      "id": "cceffa83-283e-463a-b7b1-8a6ec9337563",
      "type": "istar.AndRefinementLink",
      "source": "2bb58c75-9a35-4b6f-8e8c-85f4cb604c84",
      "target": "f0406552-00af-4442-b03d-a78e27457929"
    },
    {
      "id": "2249a9a6-06f0-43b1-9fec-3b95718424e2",
      "type": "istar.AndRefinementLink",
      "source": "51e1a122-2abe-4153-a42e-3e0c7a3bbd8e",
      "target": "2bb58c75-9a35-4b6f-8e8c-85f4cb604c84"
    },
    {
      "id": "00fb712e-aa43-4c32-b041-eacde62edc39",
      "type": "istar.AndRefinementLink",
      "source": "52781a22-2ed1-4281-9c73-e6a9d314e240",
      "target": "2bb58c75-9a35-4b6f-8e8c-85f4cb604c84"
    },
    {
      "id": "502f9ccb-1c1e-48e0-9020-dc7b5a5a9ba7",
      "type": "istar.OrRefinementLink",
      "source": "0d5629f0-4d07-46e9-a452-cac4d322e234",
      "target": "7c79330d-f564-4134-b810-47370afd7308"
    },
    {
      "id": "fa8f5bbc-5973-448f-8745-9a01d7afe960",
      "type": "istar.OrRefinementLink",
      "source": "30fdd731-24c7-424d-b77c-6b9683a76bca",
      "target": "7c79330d-f564-4134-b810-47370afd7308"
    },
    {
      "id": "7528b4c4-0d91-4e07-a9f3-ad431893b782",
      "type": "istar.AndRefinementLink",
      "source": "fc1c1d20-d10d-48d0-a717-59094a59d944",
      "target": "30fdd731-24c7-424d-b77c-6b9683a76bca"
```

```
    },
    {
      "id": "fe6524a0-4aa2-4528-84fb-117d34a4967e",
      "type": "istar.OrRefinementLink",
      "source": "0973aeef-286f-47b0-a6cf-640e43d7671f",
      "target": "fc1c1d20-d10d-48d0-a717-59094a59d944"
    },
    {
      "id": "13bb3b35-e66a-4c0d-a02f-ac8198cbc352",
      "type": "istar.OrRefinementLink",
      "source": "a219eddf-f170-454f-b20b-7568228307e0",
      "target": "fc1c1d20-d10d-48d0-a717-59094a59d944"
    },
    {
      "id": "31ba894d-707a-4670-81b3-3056559ab155",
      "type": "istar.AndRefinementLink",
      "source": "c4146a67-384f-48f8-b48a-65fbdb31c7d0",
      "target": "a219eddf-f170-454f-b20b-7568228307e0"
    },
    {
      "id": "aa89e99a-ab8d-42d6-8fad-91c599c01335",
      "type": "istar.AndRefinementLink",
      "source": "637bd788-65ab-4223-81cd-ceab64eef5fb",
      "target": "a219eddf-f170-454f-b20b-7568228307e0"
    },
    {
      "id": "2a15ced0-a60e-4a2f-8e7a-c5eb6d7ac6dd",
      "type": "istar.OrRefinementLink",
      "source": "0429a68a-713a-4918-96d9-f3fbff61569d",
      "target": "8de7a901-398e-4d37-a22d-3f954bb81601"
    },
    {
      "id": "152d6209-502b-4291-95bc-d4e2dd28adc3",
      "type": "istar.OrRefinementLink",
      "source": "f6f3d933-8b3d-4c61-893c-cca36d2972c8",
      "target": "8de7a901-398e-4d37-a22d-3f954bb81601"
    },
    {
      "id": "69b3478b-8374-4def-bdf1-1a470c1dae90",
      "type": "istar.AndRefinementLink",
      "source": "c4963052-bad5-46ef-b79e-031266546f11",
      "target": "f6f3d933-8b3d-4c61-893c-cca36d2972c8"
    },
    {
      "id": "bc91875d-7167-410e-8c18-dd86e0c77263",
      "type": "istar.AndRefinementLink",
```

```
      "source": "9c1403b9-53f3-4036-8690-3af589e5c536",
      "target": "0429a68a-713a-4918-96d9-f3fbff61569d"
    },
    {
      "id": "9603aaa6-c3e2-4699-8b2a-09800b8ae326",
      "type": "istar.AndRefinementLink",
      "source": "3867e9d3-349c-4829-8c9b-61de16ac60c2",
      "target": "51e1a122-2abe-4153-a42e-3e0c7a3bbd8e"
    },
    {
      "id": "520f7996-c674-4a95-a6fb-f2dfbc62aad9",
      "type": "istar.AndRefinementLink",
      "source": "ba3fd6ff-acbc-4a10-911a-55c4b655ed8c",
      "target": "3867e9d3-349c-4829-8c9b-61de16ac60c2"
    },
    {
      "id": "3b73c71e-d64b-428f-9144-412368798265",
      "type": "istar.AndRefinementLink",
      "source": "bbfc14b1-9da7-4112-82cf-3ec369887261",
      "target": "e3ea007b-e9df-4651-9260-8566a86c52d8"
    },
    {
      "id": "a3481701-8c90-426b-9086-0656307c3bc2",
      "type": "istar.AndRefinementLink",
      "source": "e3ea007b-e9df-4651-9260-8566a86c52d8",
      "target": "f6f3d933-8b3d-4c61-893c-cca36d2972c8"
    },
    {
      "id": "88ad5efa-86ee-44d4-9cd3-1da4e9896420",
      "type": "istar.AndRefinementLink",
      "source": "b96e97cd-d783-4f76-b006-b4c0ef93f4c9",
      "target": "e3ea007b-e9df-4651-9260-8566a86c52d8"
    },
    {
      "id": "b1c1d20d-787d-484d-b39a-70bd23f702a0",
      "type": "istar.AndRefinementLink",
      "source": "bdacda33-50aa-4850-9ce5-ad6a47368972",
      "target": "52781a22-2ed1-4281-9c73-e6a9d314e240"
    },
    {
      "id": "62c3868b-b917-436b-ad0a-37bfc1ac31b3",
      "type": "istar.AndRefinementLink",
      "source": "0c59d725-3739-47ad-8ce9-d49b65004a52",
      "target": "f0406552-00af-4442-b03d-a78e27457929"
    },
    {
```

```
      "id": "3d19403c-c0f0-4243-b017-6068a3262795",
      "type": "istar.AndRefinementLink",
      "source": "7c79330d-f564-4134-b810-47370afd7308",
      "target": "0c59d725-3739-47ad-8ce9-d49b65004a52"
    },
    {
      "id": "ab346661-81be-44dd-99e4-b7907930d12c",
      "type": "istar.AndRefinementLink",
      "source": "72f6d345-7050-4382-8593-10719959349b",
      "target": "0d5629f0-4d07-46e9-a452-cac4d322e234"
    }
  ],
}
```

Listing VII.1: Invalid input model