REASONING ABOUT CPS

USING SURROGATE SIMULATION MODELS

By

Shunxing Bao

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

for the degree of

MASTER OF SCIENCE

in

Computer Science

May, 2014

Nashville, Tennessee

Approved:

Aniruddha Gokhale, Ph.D.

Joe Porter, Ph.D.

ACKNOWLEDGMENTS

This work would not have been implemented without support from Dr. Aniruddha Gokhale and Dr. Joe Porter. I really appreciate it and feel honored to work with my two advisors. They also encouraged me and made me convince myself to continue to pursue the research road as a PhD student in Vanderbilt University.

Dr. Aniruddha Gokhale provided the motivation and idea of this research. It was abstract when I first heard of it, but the blueprint of this research is so attractive and ambitious, I devoted myself working on this project very quickly. Since the original concept is quite abstract and hard to imagine, he tried many concrete and useful examples to help me comprehend the problem first and my studying direction became clearer. Thanks to the internship in ISIS, I have known Dr. Joe Porter. He is so nice and full of knowledge, and his crazy ideas and insight on my work made me start to love studying research. He is very patient and provides me much core idea to overcome a lot of the difficulties in the whole process, and he instructed and inspired me on how to figure out the problems independently.

I am grateful to Dana, I feel so happy to be with her in these months, and the relationship established with her is one of the foremost rewards in my life. Thanks for my other friends the friends we have made in this two-year master, of course, my first friend Richard Zigler.

I love my parents, they gave me hope and opportunities to study abroad, and without their love and help I cannot be a man like I am now. I love you mom and dad.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter I

Introduction

Developing an affordable, easily accessible and scalable online CPS laboratory is significant to promote CPS education [1]. In developing such a system we are focused on a number of cyber-physical challenges including the model design and simulation strategies, and concentrate our CPS field on Reconfigurable Conveyor System.

For CPS education, it is hard to have students apply and operate a real Cyber Physical System, for instance, a real conveyor system. Concern with several challenges in real CPS industry (maybe: If we use real industrial CPS to conduct student experiments, there are several potential challenges):

1. Maximum sustainable rate: If the system design is modified can we keep the transfers going fluently without collision with the maximum rate as established before?
2. Starvation of certain paths: After reconfiguring the system, will any existing links be invalidated?
3. Prioritization: How can we set the priority for each path to ensure all packages can successfully reach the destination or transfer them with the least time?
4. Fault tolerance: If failure occurs, will the system's overall throughput be impacted?

In order to answer these questions, a model-driven approach is one of the best solutions [2]. Students can simply design their model on computer, using some basic modeling operations and complexity model logic and algorithm; the students can easily alter and modify their design to observe the performance of their system. Simulated experiments are more cost effective than expensive CPS equipment and reduce the risk of failure. All modeling and simulation work are hidden within the simulated test bed.

The paper provides a complete process to simulate the behavior of a user-design CPS conveyor system in the framework of Cyber Physical Systems Laboratory-as-a-Service (CPS-LaaS). The user-design model is sent to the background and treated offline. The extracted simulation result is finally feedback to user as an animation.

I.1 Motivation

I.1.1 Cyber-physical systems

Cyber-physical systems (CPS) pervade several application areas of societal importance, such as advanced manufacturing, transportation, health care, smart grids, and smart buildings [21]. CPS is often intelligent networks, which combine communication, computing and controllers, so that they can interact with each other and cooperate to achieve particular goals. To address the 21st century challenges, we need future scientists and engineers to be well-trained in the science and engineering of CPS. It has been amply demonstrated that problem- and project-based learning environments help students apply learned theories to solve realistic problems. Unfortunately, a vexing problem that makes it hard to support a problem- and project-based CPS learning environment stems from a general lack of access to fully equipped laboratories that can provide hands-on, practical CPS education to a large pool of future scientists and engineers.

One of the classic CPS is the conveyor system. The traditional conveyor system can transmit a bunch of packages in a section successively. The conveyor cost is relatively low, transport time is predictable, and the freight flow is stable, so they are commonly used in current logistics systems. To quote from the classic article on reconfigurable conveyor systems [4]: '*The key factor in a truly reconfigurable modular conveyor system is the ability to connect and reconnect a wide variety of modules and accessory modules that allow engineers the freedom to tweak production lines when necessary without the cost of a brand new conveyor or the risk of losing the conveyor's integrity.'* The reconfigurable conveyor system's flexibility, scalability, sustainability, and cost effectiveness can make the logistics system more productive. The performance of the system is based on the logical layout of the conveyor system. In order to observe the property more directly; it is a good idea to allow students to design and study this system to obtain knowledge of how physical and cyber systems work together.

I.1.2 Cyber Physical Systems Laboratory-as-a-Service (CPS-LaaS)

Massive Open Online Courses (MOOC) is popular in current higher education, which provides not only a high quality and free education resource, but also a complete study experience [4]. This real-time interaction platform gives us research innovative ideas, that we can also make our CPS course online and can handle as many students' system design as possible from all over the world.

To overcome this problem we have previously outlined a vision of a framework called CPS-LaaS. We surmise that CPS-LaaS will provide easy and affordable access to CPS laboratory artifacts over the Internet by virtualizing the physical CPS laboratory resources and offering them as a service in much the same way as contemporary Software-as-a-Service (SaaS) offerings in cloud computing [5]. Thus, in the CPS-LaaS vision, students are provided access to a web-based learning environment that is customized to the CPS domain they are studying. All student-led experiments are conducted using the web interfaces. The CPS-LaaS capability maps these virtualized interactions onto concrete cyber-physical resources transparently to the user, which is where the actual execution of student-led experiments takes place. Results of these experiments are relayed back to the student via the web.

Central to the success and scalability of the CPS-LaaS vision is the notion of an Analogous System [6], [7]. This analogous system is essentially a system functionally equivalent to the domain that the student is interested in studying such that it can represent the cyber and physical interactions of a class of CPS applications. For example, consider a student investigating coordination algorithms for a reconfigurable conveyor system. Since it is hard to access a laboratory with a conveyor system, it is possible to approximate the conveyor CPS by an analogous system comprising a cluster of robots whose positions and trajectories can be controlled to mimic belt movement to simulate the package transmitting flow. Such a laboratory of robots can then be used to illustrate all the behaviors of the original system. In our prospect, we plan to place the laboratory in the cloud side. So the user's conveyor system designs are simply sent to the cloud and their behaviors are simulated by those typical robots. And finally, the performance of the systems is returned back to the user's screen from web-based laboratory background.

I.2 Solution Summary

The input is a reconfigurable design from a CPS student, and the output is a behavioral simulation of this design. The solution approach has two main parts: first, for the modeling aspect, we have a complex domain-specific conveyor design are defined in the GME; second, the complex conveyor model can be mapped and transformed to the global grid, another domain-specific model, which contains only one kind of node and large dimensions so that all different species of components in the complex model are mapped to the typical nodes in the grid, and it is easy to operate and simulate the nodes in the global grid when multiple experiments are being mapped to the grid. In this thesis, we are only concerned in this scenario with one experiment. The transformation and mapping process is implemented using the Graph Rewriting and Transformation (GReAT) tool. As a background simulation, the Robocode's code is automatically generated by a GME interpreter from the global grid and is applied to generate the path logic to transmit the package, according to the package type in each input port. After acquiring the transmit speed and path, a Robocode simulation outputs the coordinate and time information to generate the Java animation. The final Java animation will be fed back to the user side to see the results of the package transmission flow.

Chapter II

Background

II.1 Cyber-Physical Systems

Although there is no unified definition for CPS, in conclusion, CPS obtains several characteristics [8], [9], [10]: 1) deeply embedded computation systems, 2) widely complex networks, 3) intelligence, 4) perception, and 5) mutual coordination. CPS is a new generation of intelligent systems, uniting integrated computing, control, and communication and also cyber and physical processes [11]. CPS provides the interface between cyber world and physical world. The cyber space can remotely operate a physical entity in secure and real-time. CPS involves system engineering of ubiquitous environmental perception, embedded computing, network communication, network control, and adds to physical systems the functions of computing, communication, precision control, remote control, and autonomous control. It is highly worth studying CPS because of its bright future and massive challenges in science and engineering.

II.2 Model-Integrated Computing

Model-Integrated-Computing (MIC) is highly-focused in modeling study; it becomes one of the most important fields in model-based software development [12]. MIC is a new approach in software development basing on modeling; it provides a very flexible and multi-dimension modeling framework. It not only has the characteristics of traditional software modeling, but also has its own features in embedded software development. The model is located in the center of the complete system life cycle. It effects the whole development process, and involves modeling analysis, verification, integration and maintenance. It is very important in modeling transformation; it has different representation in different levels so that fit for any form of analysis, verification and simulation tools. In MIC, we use meta-languages to represent the key components of the system information by modeling. Besides, this language can create a system environment to simulate physical conditions.

II.3 GME

The Generic Modeling Environment (GME) is a customizable generic modeling environment, which supports metamodel modeling and Program Synthesis [13]. It applies UML class diagrams to describe a domain-specific modeling language [15]. The reason GME is customizable is that it can apply a meta language to describe the concepts, relationships, and constraints of specific domains. GME also provides an interpreter mechanism to construct the model, which can be used to automatically map the system model to executable code.

II.4 Graph Rewriting and Transformation

Graph Rewriting and Transformation (GReAT) is a transformation tool based on graph models, which can be seen as a domain-specific language in the GME modeling environment using meta-language defined in GME [14]. GReAT has an engine GReAT-E (GE) that is used to describe model transfer rules and graph rewriting and transformation language [16]. GReAT uses Pattern matching input to the appointed input metamodel and output metamodel. So that when there is an operation applied on the input metamodel's instance, it can solely effect on the output metamodel's instance, which in this way a bonded connection has been established.   In the rules of GReAT, the finest unit of model transformation definition rule is an atomic rule, and each rule specifies the rule input and rule output. The mapping relationship of the input and output model is defined by the Attribute Mapping object. It can set the output model's attributes according to the definition of transformation semantics and the input model's attributes. In an atomic rule we can also define the operations of model creation, deletion, and attribute modification. Users can define a series of atomic rules to design the transformation process. GE is GReAT's executing engine which is a model interpreter built using the Model interpreter framework provided by GME.

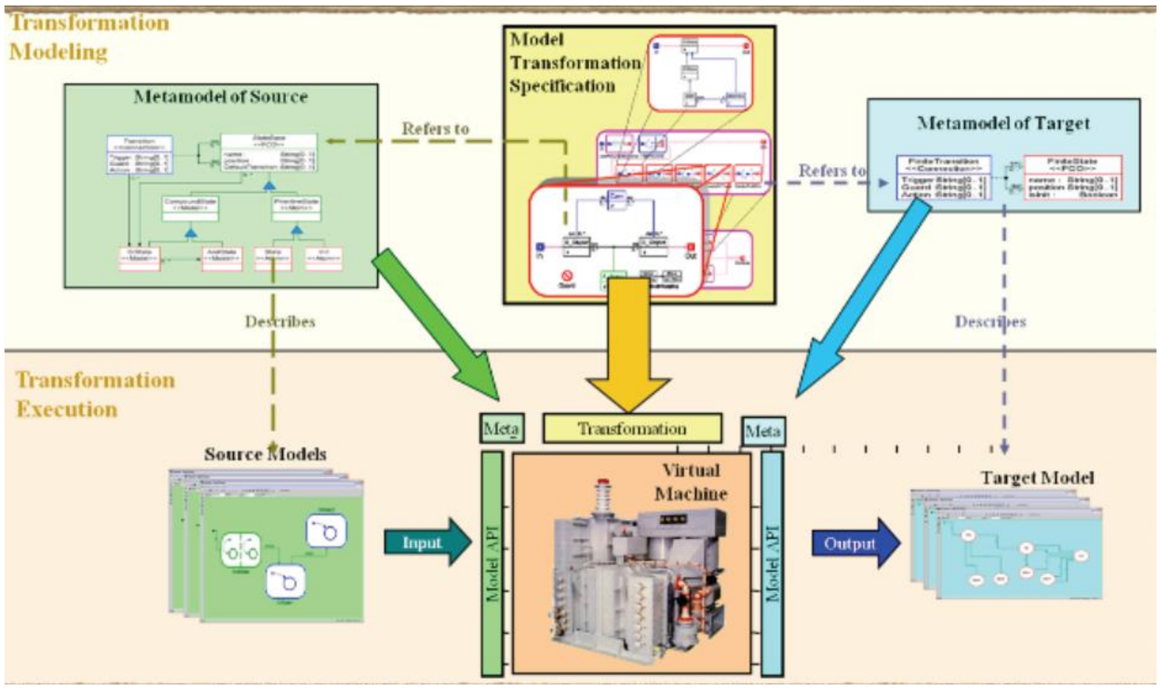

Figure II.1 The approach Used in GReAT

II.5 Robocode

Robocode is a tank combat simulator developed by IBM Alphaworks [17]. The tank must avoid attack by other tanks, whether the tank is in enemy team or the same team, the tank will lose energy when it is hit. The tank should also avoid running into walls or colliding with other tanks. The objective of the battle is to shoot and destroy the opponent's tanks

It is hard to imagine and associate Robocode with a Conveyor System. However, there are several properties very interesting to be applied to simulate a conveyor system.

1. Each tank robot contains noiseless radar, which can sense the tank collision, battle frontier, tank hitting event, tank appearance and on-going [18]. So the tank itself can be treated as a sensor to monitor the event in the battle field, and the radar can play a role as the conveyor system's sensor.
2. Within a tank team, each teammate can send and receive messages in a variety of formats. According to the message content and type, the typical command can be conveyed to each tank to simulate the actions of the conveyor system. Hence message requests and responses within a team can play a role as signal communication.
3. Because of the nature of battle, each tank can set its own bullet power, and then the bullet power can act as an identifier for the different packages in order to distinguish the package type. In this way the bullet can simulate the behavior of the physical feature. The velocity of a flying bullet is defined by the equation [19]:

*Bullet Velocity = 20 – 3 * firepower*

4. Robocode is a Java-based code, we take advantage of Java’s cross-platform independence properties so that the Robocode simulation can be used anywhere. And as it is also a real-time running and on-screen program, we can easily debug and make the logic simulation visualize. As the background simulation, the user does not know what is happening in Robocode. However, due to Robocode’s own file writer function, we can take advantage of those functions to extract the location and time information when running battles to generate the final animation.

Chapter III

Implementation

III.1 Basic Sample Model to Simulate

In this thesis, our approach is based on the example model logic in Figure III.1. There are four main components in the Conveyor system. The input ports are the places that receive the packages, and the packages will be finally sent to the output ports. The packages are transmitted by conveyor belt, which can move the parcels from belt's one end to the other end. In our system design, we name the belt as segment, which is a vivid metaphor. Each turnaround can connect multiple segments, it act as a transmit station. When a package is transferred to a turnaround, the turnaround will transmit the package to the segment that needs to go to. Three Input ports I1, I2, I3 and Three Output ports O1, O2, O3 are in this design, correspondingly we categorize the package type as **Big, Medium and Small** can be transferred from input to output ports. S1, S2… S13 are segments acting as the belt to move the parcel. T1, T2….T6 are six turnarounds used to switch the package.

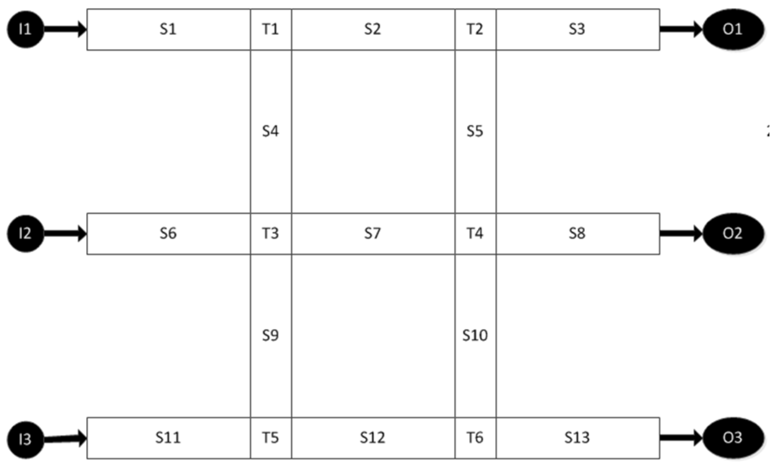

Figure III.1 A sample Conveyor System Logic Design

III.1.1 Overview of the Solution Architecture

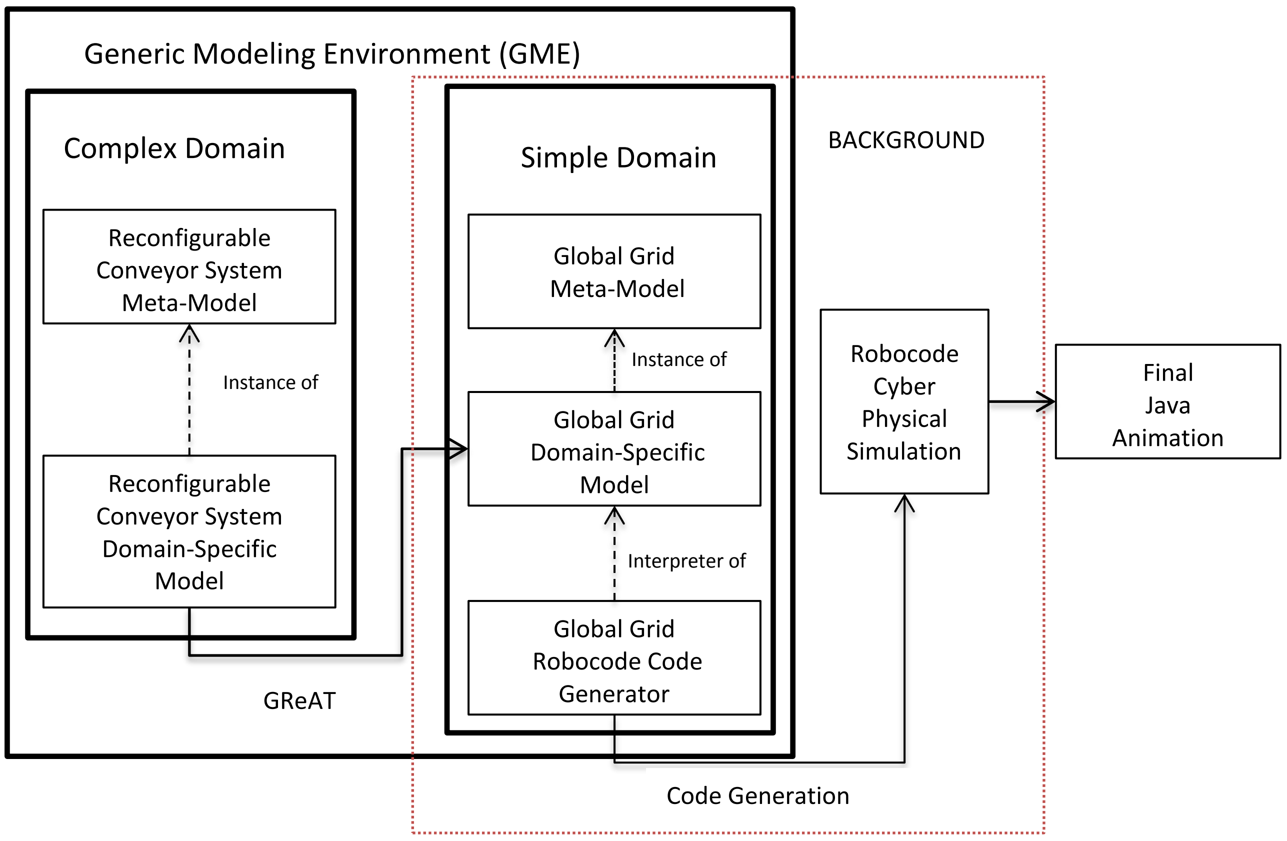
Generic Modeling Environment (GME)
Complex Domain
Reconfigurable Conveyor System Meta-Model
Instance of
Reconfigurable Conveyor System Domain-Specific Model
Simple Domain
Global Grid Meta-Model
Instance of
Global Grid Domain-Specific Model
Interpreter of
Global Grid Robocode Code Generator
GReAT
BACKGROUND
Robocode Cyber Physical Simulation
Final Java Animation
Code Generation

Figure III.2 Overview of the Solution Architecture

According to the Figure III.2, in modeling part, there are two domains, the Complex Domain is used for user to design their own design, and their design can transform to a form defined in simple-domain, the **Global Grid**.

The idea for global grid comes up with the prospect of mapping as much conveyor experiments as possible (In Figure III.3, experiment E1, E2, E3 are mapped from user side to grid, reserving a corresponding space in the grid to simulate their system design, as we mentioned before. However, in this paper we only consider the scenario mapping one experiment to the grid). Because there are multiple component models in the user-design side, for instance, input port, output port, turnaround, and segment, we can hardly design a panel with horizontal and vertical path to dynamically generate the corresponding nodes, and the size required to create such a panel is hard to calculate. Moreover, in this research the only specific domain we concerned with is the conveyor, if we design and apply another CPS domain, the dynamic panel with multiple nodes to fit for mapping a great number of nodes from CPS student is unreasonable. We hence design a global grid; this is a kind of panel with only one kind of node and necessary connections.

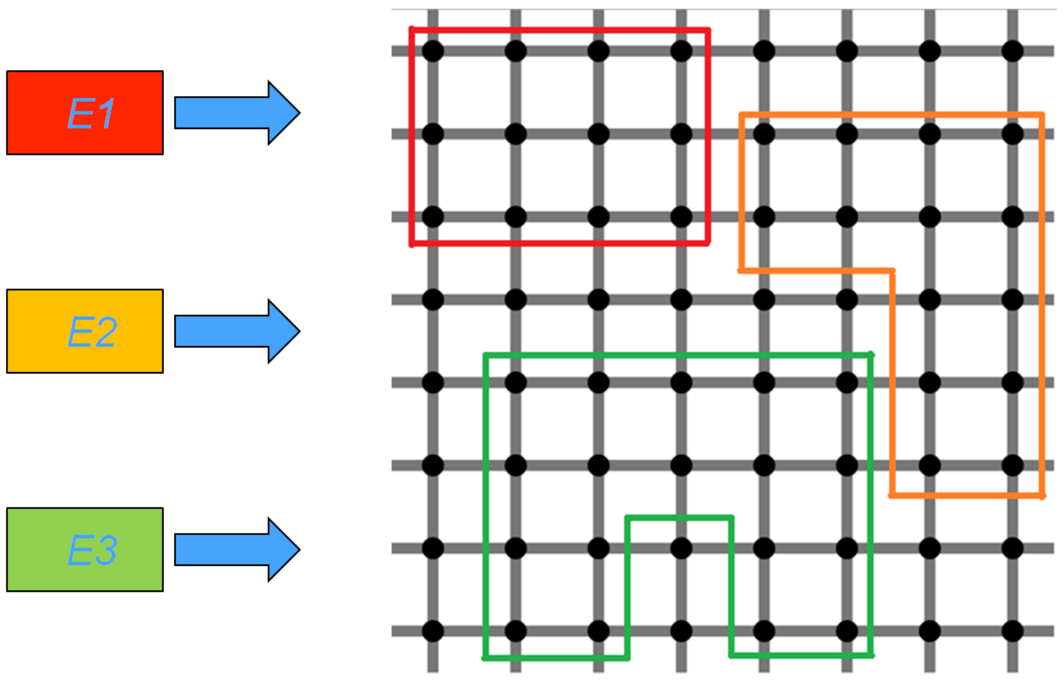

Figure III.3 A sample Scenario of Mapping three experiments to global grid

The Grid can generate the code for the Robocode to simulate in the background, which the user will not know what happen in the background, what they will get is a final java animation to see their system performance.

III.2 Reconfigurable Conveyor System Design

III.2.1 Complex-domain Metamodel

The metamodel is presented in Figure III.4. In this metamodel, the highest-level model is **Experiment**. It contains **testSystem, Input and Output.** Model **testSystem** is a reference of model **System**, so we can treat the **testSystem** is holding the same properties with **System**, and all operations to the **testSystem** are actually effected in the corresponding **System.**

System contains three different kinds of models, **Input**, **Output** and **Block**, these three models are the key component to build the Conveyor system. **Package** is contained in **Input,** which is used to represent different kinds of parcels need to be transferred in a conveyor system. **Input** and **Output** models are intuitively used to design the input ports and output port of a conveyor system; model **Block** is inheritance by three child-model **SegmentWE**, **SegmentNS** and **Turnaround**, which the first two of three child indicate the conveyor's belt moving from west-east direction, the belt moving from north-south direction. Turnaround is a transit point, acting as the connection between one belt and another belt segment to keep or alter the package's original direction as we disscuesed in the basic sample model. Three connections are defined in **System**. **BlockToBlock** makes connections among **Blocks. InputToBlock** creates the connection from **Input** to **Block** since the **Input** in system does not have inward link**.** With the similar principle, **BlockToOutput** builds the connection from **Block** to **Output** because of **Output** cannot have outward link.

The Attribute in each model is quite straightforward. **NodeType** in block is used to distinguish the Meta type of each node, which will be applied as a directly link to map to the global grid. **PackageType** is defined for package type. **Speed** is the belt transmission speed, it can be random set with the number greater than zero, however, in this paper, we set three default speed to compatible with three different packages, which the belt's speed is decided by the package type.

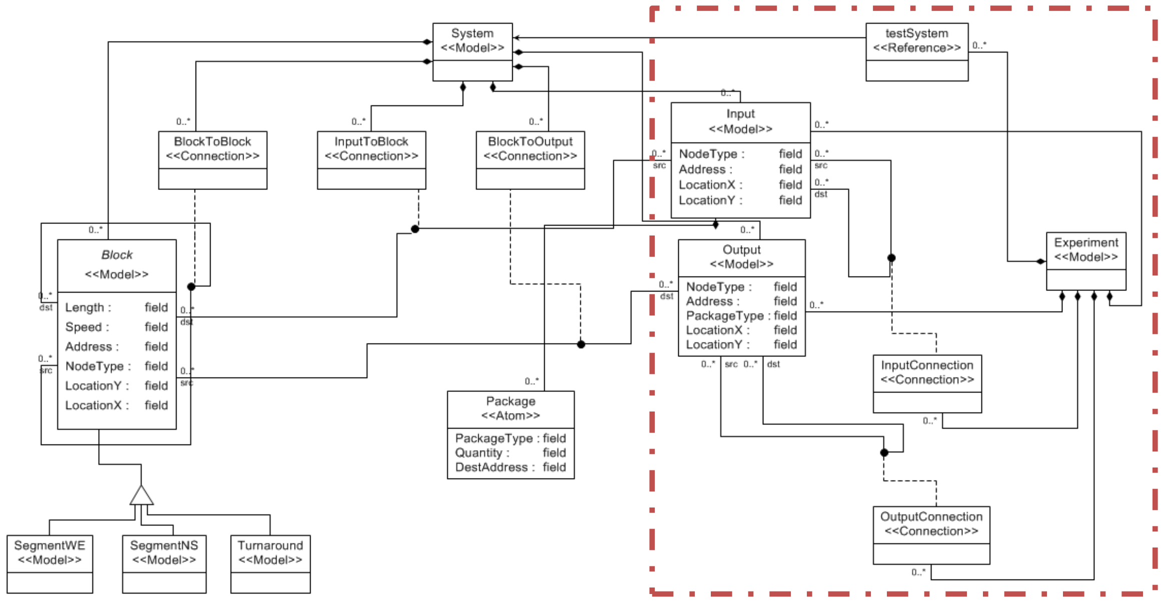

Figure III.4 Reconfigurable Conveyor System Metamodel

In the higher hierarchy (please see the red region in Figure III.4), **System** is referenced as the reference **testSystem** and embodied in the **Experiment**. Meanwhile, **Input** and **Output** are also contained in **Experiment. InputConnection** and **OutputConnection** allows that the model **Input**/**Output** designed in **Expreiment** can connect with the **Input/Output** designed in **testSystem.** According to this connectivity, we can activate any number of input ports and output ports to transfer and receive the package, it will greatly generate more permutation and combination of experiments using only one conveyor system design, and every experiments can be mapped to the global grid, this process will be discussed in the future work on scheduling multiple experiments to grid.

III.2.2 Complex Domain-specific Model

A domain-specific model is an instance of its direct metamodel. Figure III.5 is a domain-specific model which is based on Complex-domain metamodel defined above there are 3 **Input** models, 3 **Output** models, 9 **SegmentWE** models, 4 **SegmentNS** models, and 6 **Turnaround** models with related connection. The architecture is based on the initial model design in Figure III.1. One-end-arrow represents the package interflow of goods and materials in the conveyor system. Although the connection lines within Segement and Turnaround have no arrows, we acquiesce the horizontal flow direction is parallel with arrow way.

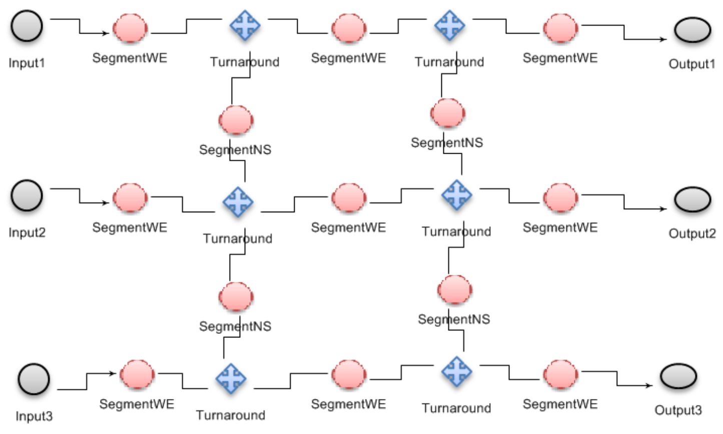

Figure III.5 Domain-specific model – NewSystem

According to Basic sample system main canvas

In the view of **Experiment** model (Figure III.6), namely NewExperiment, NewSystem is presented as a reference in the center, each input ports are connected with one Input model, and this is the only way to represent that the NewSystem’s input port can be activated. If there is no connection within the input ports and outside models, the input ports of NewSystem will still be mapped to the on-going process, however, the node after mapped is no longer activated, namely it is invalid to use in simulation part, this will be discussed in the description of the GReAT transformation. In this case, all input ports are activated and going to be mapped, meanwhile all output ports are all activated with the same meaning. Eventually, when transforming this conveyor system design to the global grid, all activated ports, turnarounds and connections will be mapped and valid.

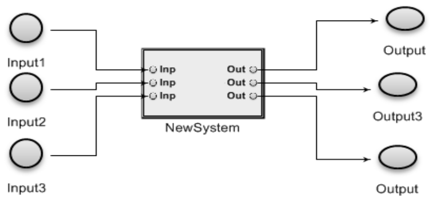

Figure III.6 Top hierarchy of the NewSystem

III.3 Global Grid Design

III.3.1 Simple-domain Metamodel

In Figure III.7, the uppermost level of the meta model is **Grid**, and it contains only one single type of model, namely **Node**. There are two reduction strategies. The first one is within the attributes of **Node**, all attributes in the complex domain’s model **Input, Output and Turnaround** are wrapper in the simple domain’s model **Node**. The second one is that we simplify the model **SegmentWE** and **SegmentNS** as just a connection, namely the connection **EastToWestConnection** and **SouthToNorthConnection** as well as adding the related attributes. In this way in the grid, we can simply consider the behavior of the connection and it is easy to modify to fit for other need of different CPS domain. Since the **Node** can play a role with **Turnaround**, we set 4 models as the ports of **Node**, the model **East**, **West**, **North** and **South**. **East** can only connect with **West**, and **Noth** can only connect with **South**, in this case it ensure the direction of the grid fairness and make the whole grid as a vertical and horizontal panel. Altogether **Node** can represent the three key models in the complex domain, and the connection in the grid can in a role of conveyor belt.

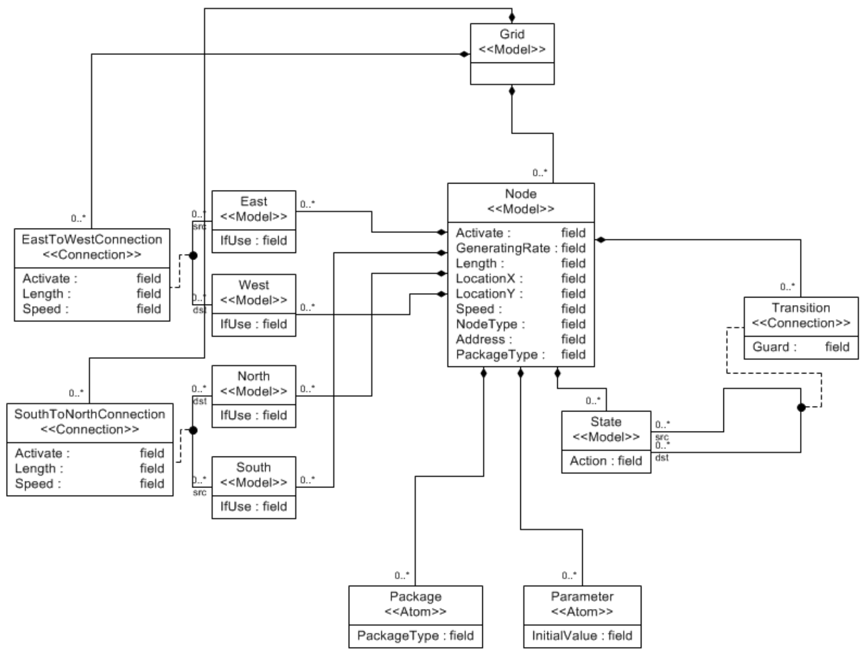

Figure III.7 Global Grid Meta Model

III.3.2 Simple-domain-specific model

The global grid instance is quite comprehensible, according to the DSML model of complex domain, 3 input ports, 3 output ports and 6 turnarounds will be mapped to as the node in the grid, the segment belts are mapped as the connection within each node. The result presents in the Figure III.8 In the next part, we will present the detail of how this mapping transformation works.

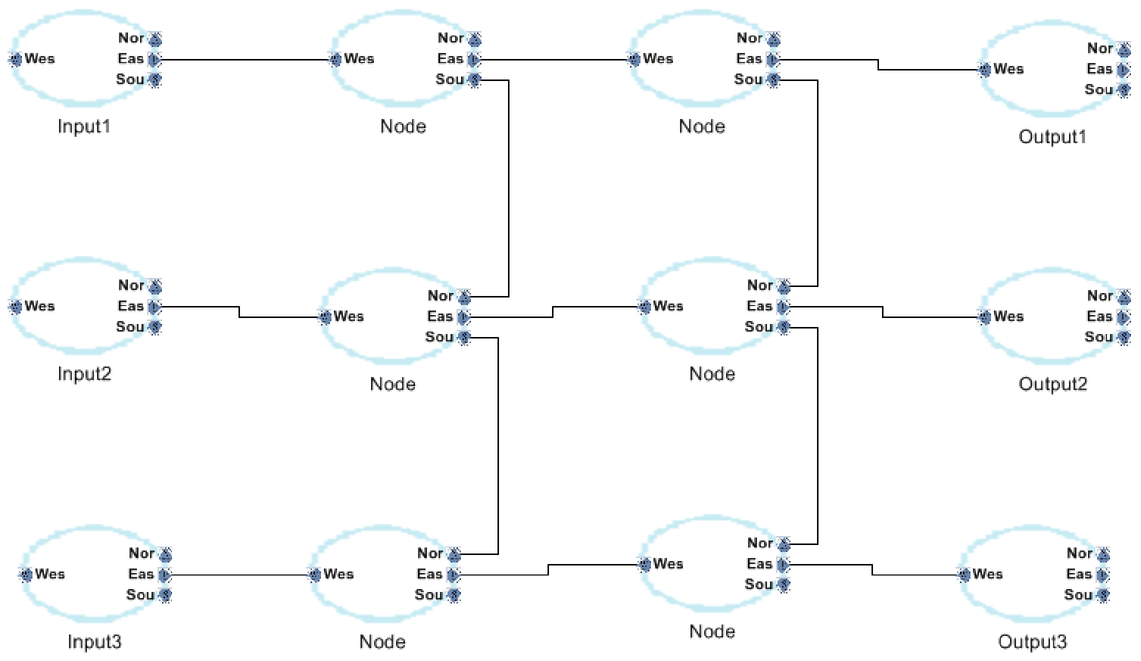

Figure III.8 Global-Grid Domain-specific Model

III.4 Graph Rewriting and Transformation

GReAT is software to rewrite and transform the current GME DSML model to a new one, whether create a new DSML model or refine the current one. In this paper, we map the current conveyor design to a new grid, because we only need to simulate one experiment indeed. In next step, we will map the multiple current conveyor design experiment to the huge grid panel; this will be discussed in the future work section.

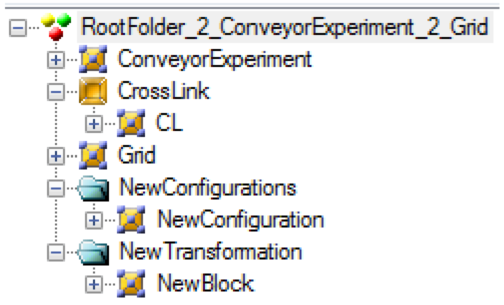

Figure III.9 GReAT Working Structure

In Figure III.9, we present the structure of our GReAT's work. We first need to import two GME meta model, one is the source meta model **ConveyorExperiment**, the other one is the destination **Grid**. Because the transformation definition by GReAT is based on the meta language to create the basic level association between two models, we do not need the DSML model as it is one of the instance of the meta model, therefore not representative. However, the DSML model is applied in the real transformation, the ultimate goal is making the transformation on a DSML model. **Crosslink** defines the inner association by an **Association Relationship-type connection** to link the source meta model's component to the destination one, so that the components lying on the two sides of the link can completely bound with each other. For instance, after reading a sample DSML conveyor model and an **Input** model, create a new **Node** in the grid, since we have defined the cross link (please **see Figure** crosslink in Figure III.10), then all the operation on the read Input port will also effect and only effect the newly-built Node itself. Otherwise we would have to build a separate table and search for the corresponding node in each rule.

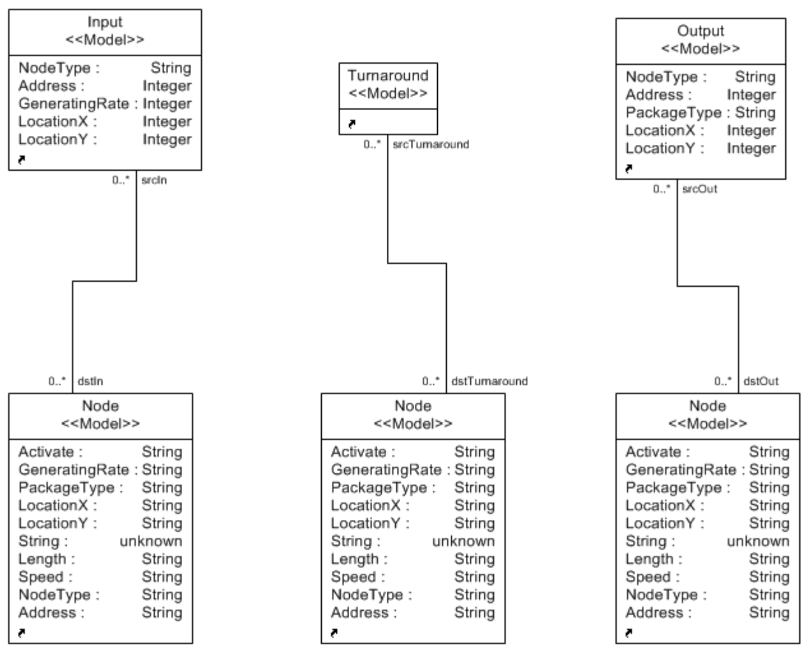

Figure III.10 Details of the CrossLinks

NewConfiguration defines the concrete source file to be rewritten and transformed as well as the output destination file, we also need to set the corresponding meta model prototype define in which level GReAT should start to read and create the files. In our case, we all start from RootFolder, this is the top level in each DSML model. **NewBlock** contains the transformation logic; as follows Table III.1 presents the four main creation logic strategies' source components and destination components.

Table III.1 Components participating in the transformation in each step

Step	Source	Destination
1	A system design	A new grid panel

2	Input ports;
	Output ports;
	Turnaround.
		Node(East, West,
		North, South ports)

3	Input ports, output ports, turnaround	Node.East, Node.West, Node.North, Node.South
4	Experiment with System's reference	Change the corresponding Node's attribute.

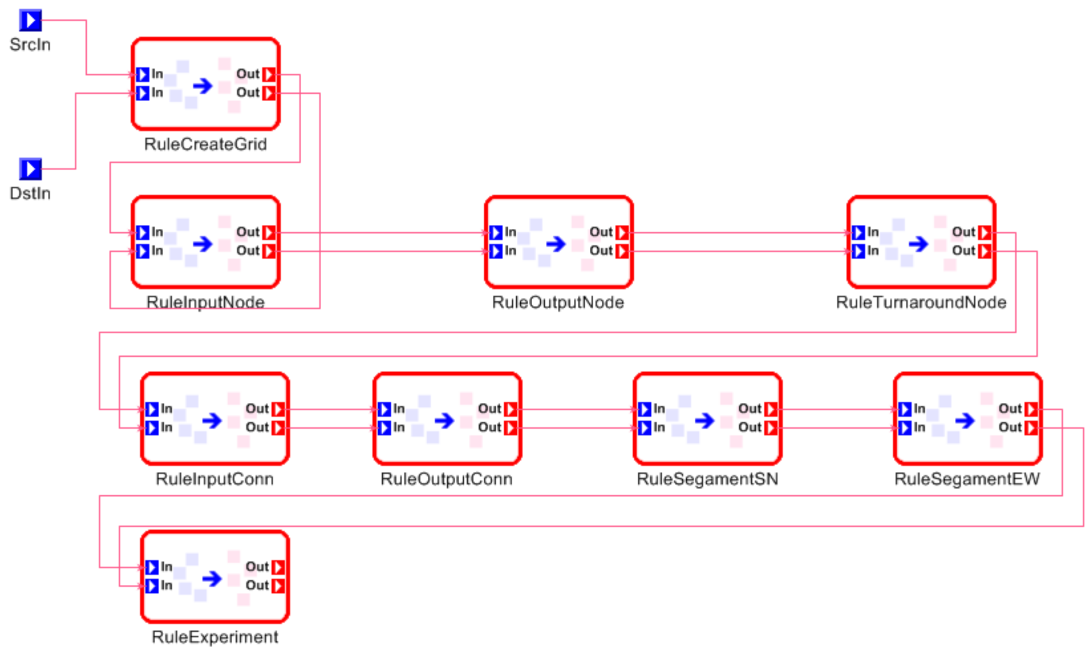

Figure III.11 Rewriting and transformation flow defined in NewBlock

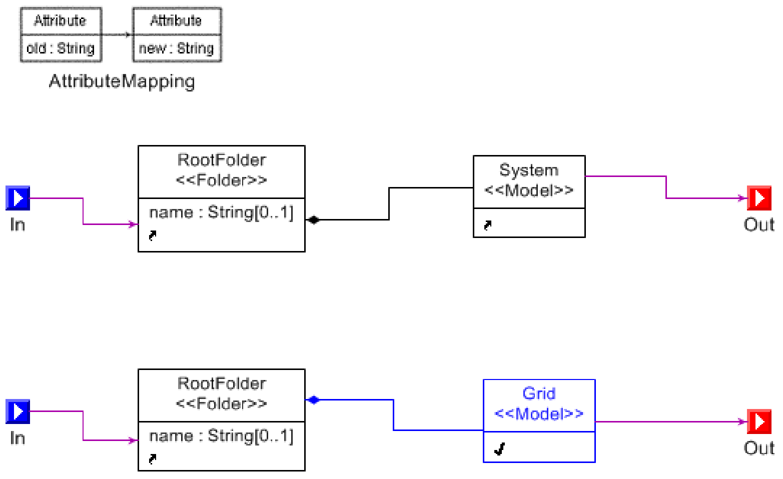

Figure III.11(a) Detail of RuleCreateGrid

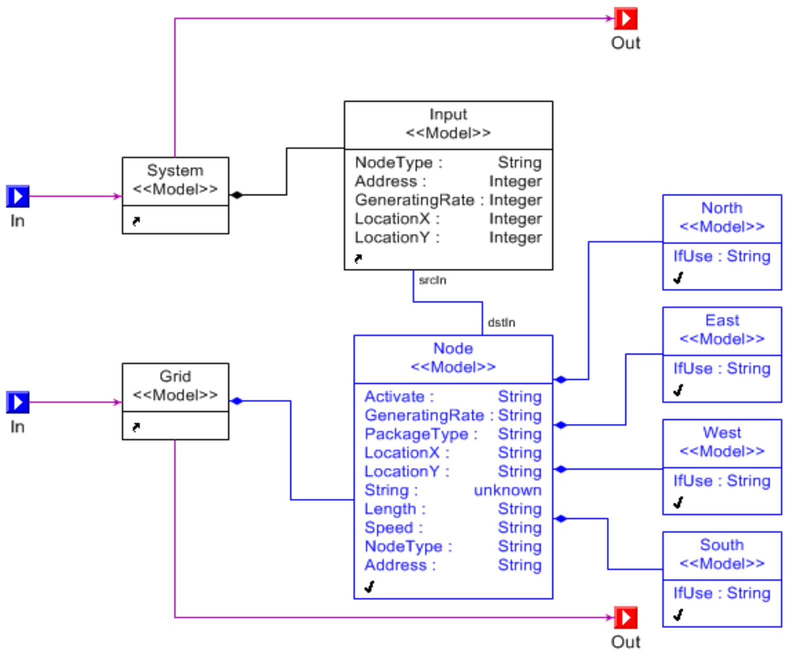

Figure III.11 (b) Detail of RuleInputNode

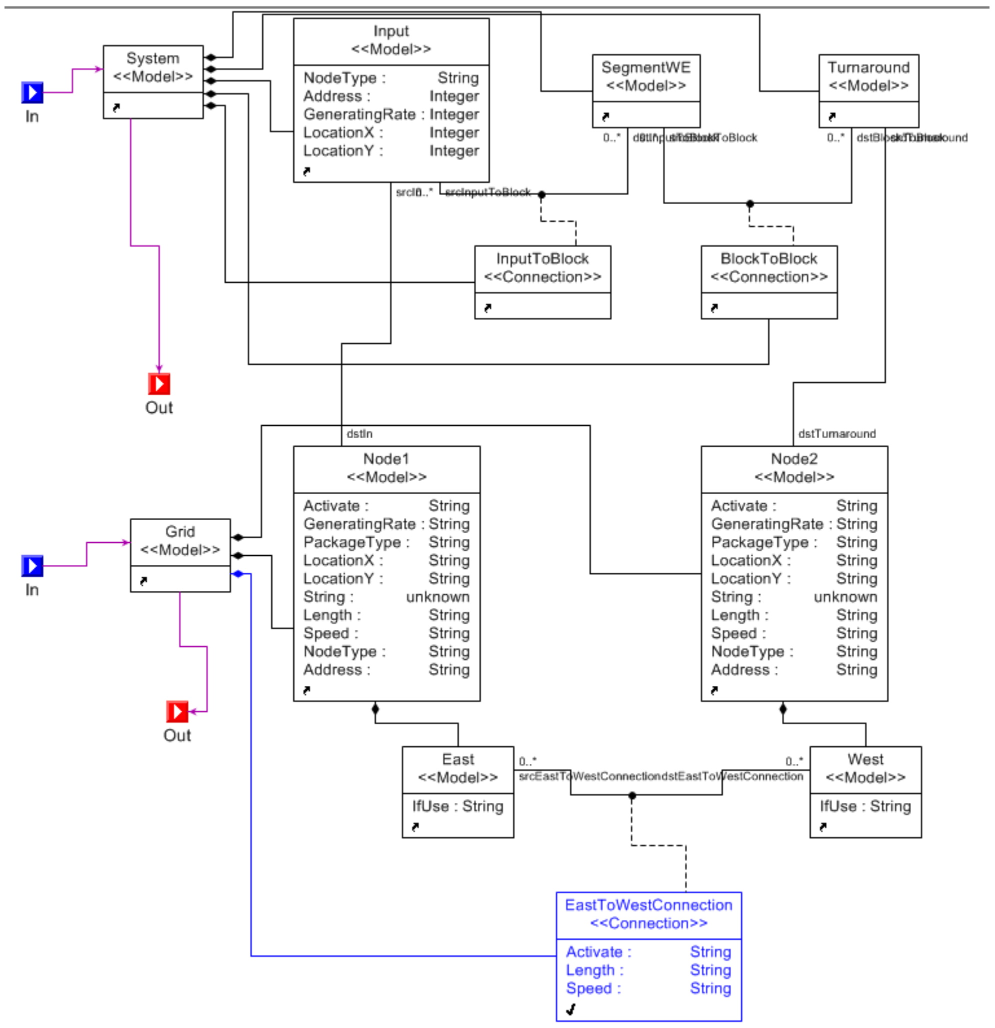

Figure III.11 (c) Detail of RuleInputConn

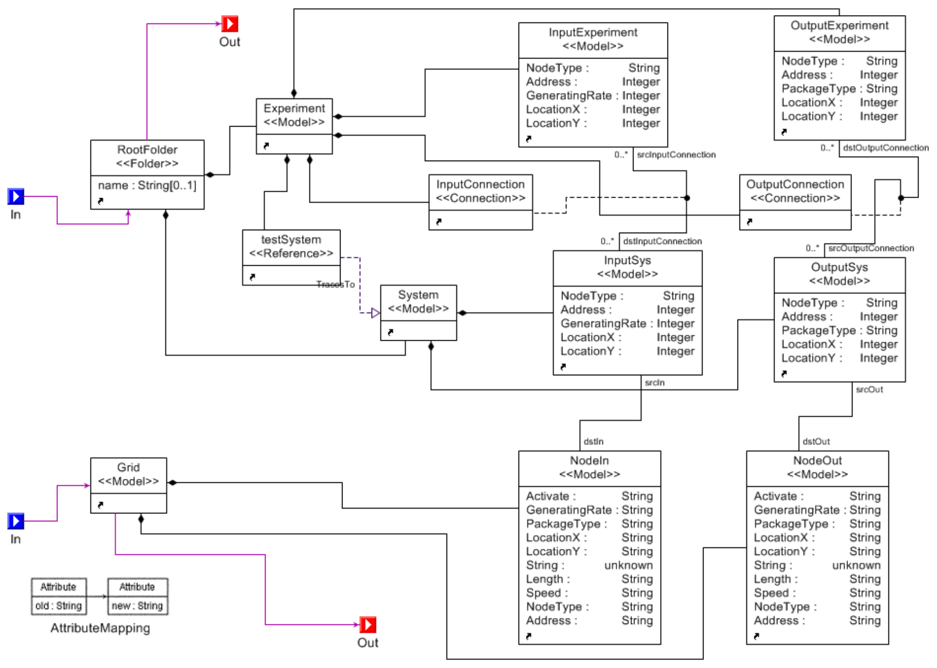

Figure III.11 (d) Detail of RuleExperiment

Figure III.11 presents the overall structure of the four steps, and III.11 (a, b, c, d) present the sample details in each step. Step 1 creates a new grid panel after reading a system design based on the complex domain. Step 2 Create Node in the grid and make the **src-dst pattern association** which is a representation of the Association in the **CrossLink**. So whether the source inputs are model **Input**, **Output** or **Turnaround**, they will only be mapped to a typical node. In Step 3 we create a Node Connection. Because in step 2 we have already created the model pattern association, this step will be mapped the **segment (WE or NS)** from the source side to act as connection of Nodes' corresponding ports in grid. Step4 activate the node in grid represent to the input & output port. After the four steps above, the DSML model in Figure III.5 will be transformed to the model shown in Figure III.8**.**

III.5 GME Interpreter

A GME interpreter is used to extract the necessary information, and the extraction logic can be written in C++ in order to create the required code. This process is automatic through generative capabilities. This procedure also involves CTemplate. CTemplate is a simple but powerful template language for C++. It mainly focuses on separating presentation logic from application logic [20]. The ctemplate script will be embedded into a ".tpl" file. Once we start the interpreter, the template file can be filed with code, and the code is created by the information from the grid. For example, the tank location according to the component's coordination in grid, the tank type according to the logic from grid, the battle time needs to be simulated. The detail of the related tank issue will be discussed in next section. Finally we generate and output our destination file -- the Robocode's '.battle' file, which records the tanks' type, initial location (the tank's position of the battle field), initial heading direction (the heading decides the shooting direction) and initial rador direction (the direction of sensor area. In this paper, we set the radar direction is the same with heading direction). In the next part we will present the detail of the tank-design.

III.5 Robocode

III.5.1 Tank Design Description

In order to simulate the behavior of the whole conveyor system, we design multiple tanks to simulate behavior for the model in the complex domain. In Table III.2 exhibits all tank types we have designed. Figure III.12 shows the initial screen of the Robocode simulation.

Table III.2 Robocode Tank Type and Description

Tank Name	Description
tank-Input	Act as the input port.
tank-Output	Act as the output port.
tank-Conveyor	Act as the conveyor system's belt (segment).
tank-Intersection	Act as the turnaround of the system.
TeamLeader	The controller manager and the principal monitor.
tank-FakeInput	No exact meaning. The Robocode is a battle game, we have to define at least two opponent side to ensure the simulation work. The FakeInput is treated as a common and only enemy with all the other tanks. The FakeInput is simply locates in a default place in the battle field, without any cyber or physical behavior, its appearance is used to make the whole simulation procedure keep working. Meanwhile all the rest of the tanks are set as a team, and within the team, each tanks can communicate with each other applying message sending and receiving.

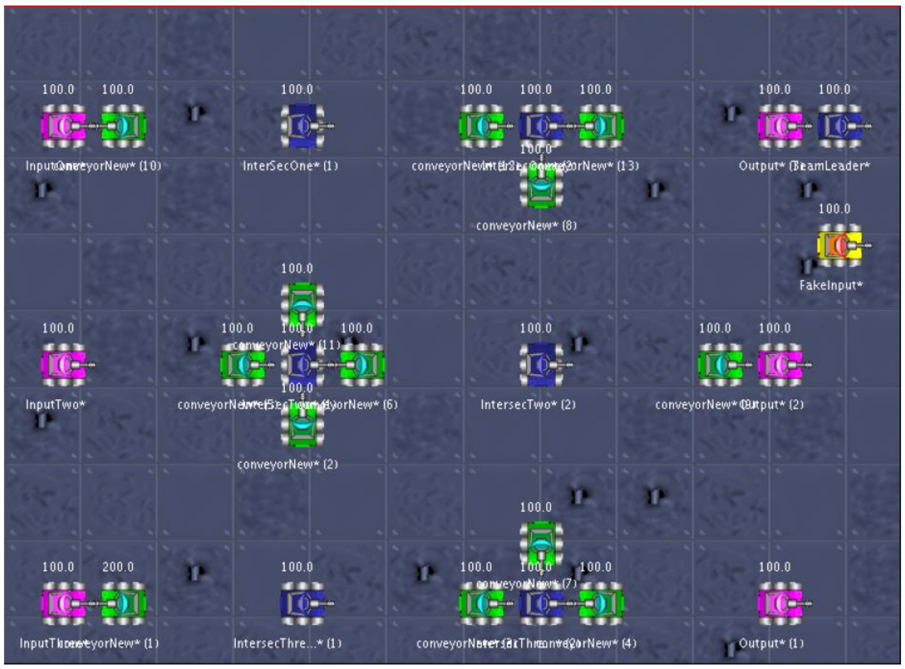

Figure III.12 Initial Screen for the Robocode Simulation

based on the basic example model

Here is the detail of each tank.

1. tank-Input

   This tank simulates the behavior for the input port. It can receive any package with small size, medium, or large. It can directly receive commands only from the team leader. There are two command types, one is to start calculating the package delivering safety path, and the other one is starting to transmit the package. The radar is applied by the tank. The heading of this tank is unchangeable.

2. tank-Output

In our case, three different kinds of packages should be sent: Large, medium and small. Tank OutputOne receives small packages from the input ports, tank OuputTwo receives medium packages, and OuputThree receive the large ones. It can receive the message from its nearest conveyor tank. It also has communication with the team leader before transmission and complete transmission each round. The radar is not inactivated for this tank.

3. tank-Conveyor

Tank-conveyor performs the belt movement in the conveyor system. It represents the direction of the conveyor movement, and it is irrelevant with the real package transmission. It can communicate the message from the nearest Input tank or Intersection tank. The radar is activated to monitor the Intersection it faces. The heading of this tank is either towards the Intersection or towards the Input/Output Tank.

4. tank-Intersection

Tank-Intersection is acting as the Turnaround to transmit the package. It can receive the message only from the conveyorNew tank if the tank faces the turnaround, and sends its message to the conveyor tank it faces. So the package transition direction is defined by the heading of the Intersection tank. The radar is activated to capture the conveyor tank. The heading of the tank-Intersection runs along horizontal or vertical lines, facing the Input/Output ports or another intersection.

5. Teamleader

The teamleader is the commander of the team. In the real conveyor system, the industry applies Radio Frequency Identification (RFID) [22]. There is a tiny silicon computer chip and an antenna in each package. Remote radar can scan and send the information of the package to a database. Then the CPU can analyze the parcel’s information and schedule the transferring mission. In the simulation, the teamleader is the commander of the team. The inner logic and the total number of input & output ports of the conveyor system design select the type of the teamleader. The definition of package transfer logic design and decision are embedded in it. Three main functions are contributed by teamleader. Firstly, teamleader considers which input ports should be activated to send the package. Secondly, the teamleader sends a direct command message to the activated input ports to generate the safety path, which will be explained in next section. Finally, according to the response of the corresponding tank-output, the teamleader transmits commands to the activated input ports to begin the package transfer. The teamleader is selected according to the number of input ports as well as the output ports.

III.5.2 Safety Path

The existence of a safety path makes it unnecessary to consider fault tolerance.

Definition: The safety path is a logical link from one of the input ports to the output ports. In order to ensure the safety without stopping, two steps are used:

III.5.2.1 Activate the Input ports

Within the path, only one package can be transmitted, so no collision is allowed to happen in it. When transferring the package, no other package from a different input port can share this path. Until the package is successfully transmitted from the source to the destination, the path stays in a busy state. This is implemented by a brute force algorithm: First of all, the tank-teamleader acquires the map of the following package type of each input ports. The tank-input port is indexed by a natural number, so the lower the number of the input ports, the higher transmission priority is given. Traverse the input ports from lowest to highest index -- if the package needing to be transferred will cross several lines, then the input ports located in the involved lines will not be activated. If the package transferred from any other input ports will cross the involved lines, then that corresponding input port will also not be activated. After traversing all the input ports from the lowest index to the highest the first time, begin to create the safety path for this round.

Pseudo code:

```
Start scan from Input_i(i=1,2,...) -> the package will be sent to Output_x(x>=i).
   activate Input_i and start to scan Input_(x+1).
         If Input_(x+1) -> the package will be sent to Output_y (y<=x),
                  then skip Input_(x+1) and start to scan Input_(x+2)
                           scan traverse
         Else
```

activate Input_(x+1) and start to scan Input_(y+1).

scan traverse

III.5.2.2 Create the safety path

After activating the input ports, we divide the logical design into several sections, and in each section, we apply the shortest path to transfer the package, for instance, as the Figure III.13 exhibits as follows: for Input_1 to Output_1, there will be only one path, and no back edge (direction west) to ensure the shortest path. For transferring from Input_3 to Output_5, the path direction can only be east and south. The Input_8 to Output_7 path exhibits the same principle.

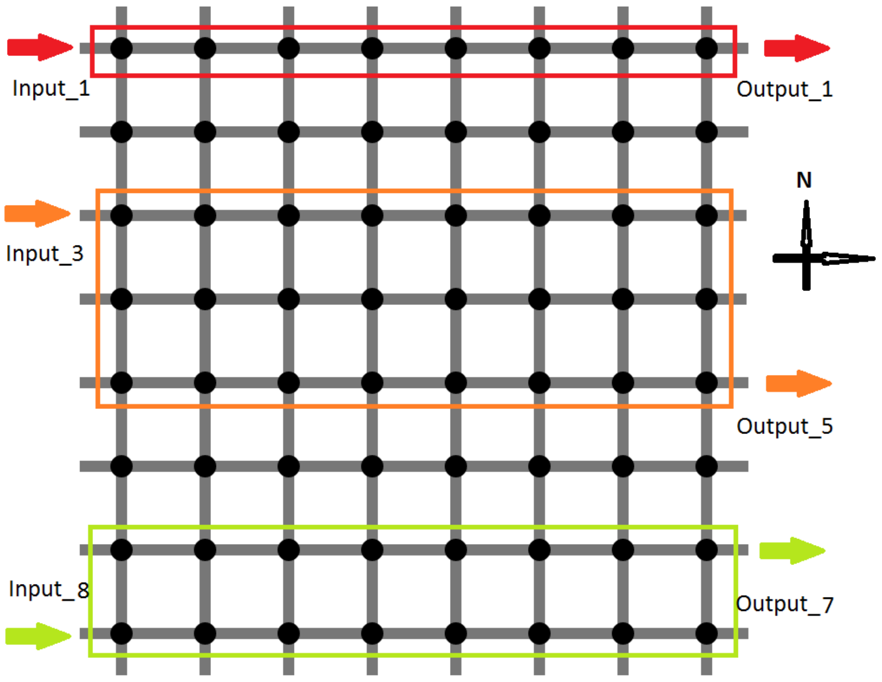

Figure III.13 A sample Grid Path Activate Status

III.5.2.3 Cyber & Physical Simulation Feature

The procedure to create the safety path is this: First, after being activated by teamleader, the tank-Input scans its nearest tank-conveyor and sends a message to it, so that the tank-conveyor will stop right next to the tank-Input. In this way, it simulates the scenario that this section of belt is waiting for the in-coming package. Besides, the tank-Input will send the package information to the facing tank-intersection. After that, the tank-intersection who received the package size message will turn its heading according to the value of the y-coordinate. The pseudo code presents how to compare the coordinate y-value with current tank-Intersection and the destination output port.

```
Final_dir = The direction of y value of the destination output port.
current_dir = The current tank-Intersection's y value
if( current_dir == Final_dir )
    keep the heading to the East
if( current_dir < Final_dir )
    if there is no tank-Intersection on East
        turn the heading to the South
    else
        random decide the direction in(South,East)
if( current_dir > Final_dir )
    if there is no tank-Intersection on West
        turn the heading to the South
    else
        random decide the direction in(South,West)
```

After the heading is decided, the tank-Intersection scans its nearest tank-Conveyor, asking it to stop and wait next to the tank-Intersection itself. The principle here is the same with tank-Input, namely make sure the conveyor belt is waiting for the package. Repeat this process until the tank-Intersection has found the final destination tank-Output, after which a safety path has been built. The tank-Output then sends a message to teamleader to convey the ready status.

Finally, the teamleader accumulates the numbers of received ready-status messages, till all safety paths have been established. Altogether, the creation of safety paths for this round is fully completed. In Figure III.14, there is a safety path from top-left corner's tank-Input to bottom-right tank-Output.

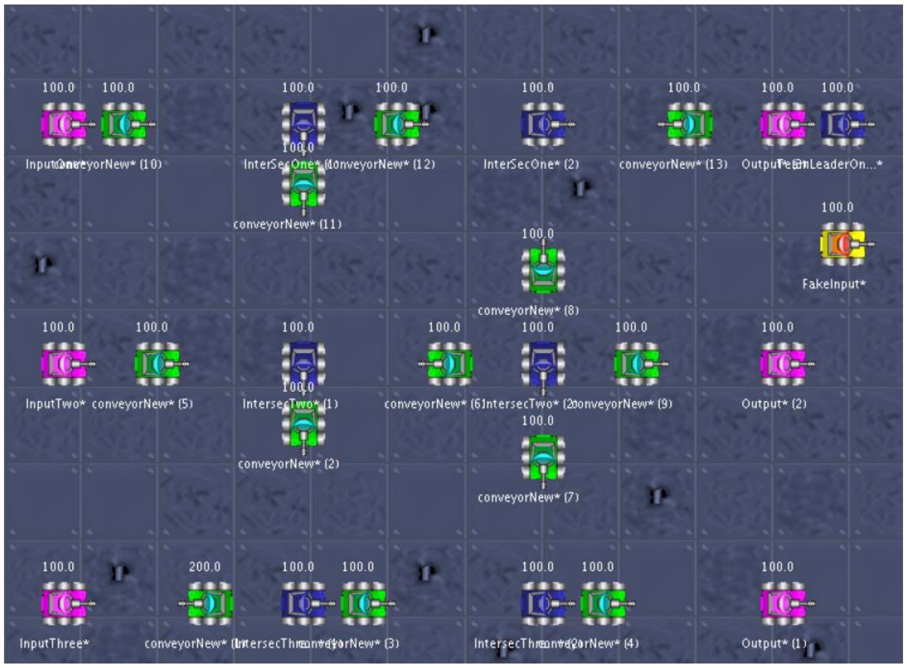

Figure III.14 An illustration of safety path

III.5.3 Package Transmission Simulation

The transmission order is given by teamleader right after the all safety paths have been built in each round. In Figure III.15 illustrates the abstract bullet shooting sequence in one safety path.

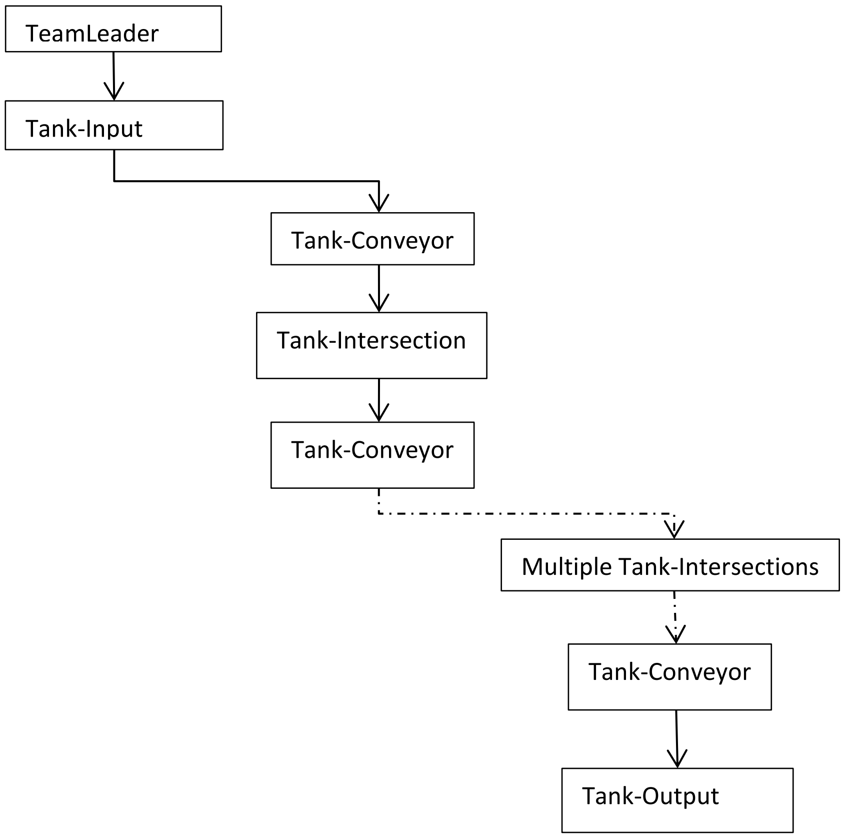

Figure III.15 The Shooting Sequence in a Safety Path

III.5.3.1 Bullet shooting

In Robocode, we set different bullet power to represent different package sizes. As one of the typical battle events in Robocode, we define two on-hit events to treat the situation when the tank is under attack. Table III.3 shows the detail of those on-hit events for different tanks. All transmitting happens from the tank-Inputs, when they shoot the bullet with established power under the order issued by teamleader.

Table III.3 Tank On-hit event Description

Tank Type	On-hit event
Tanks-Conveyor	The tank is hit by tank-Input or tank-Intersection, according to the hitting damage, calculating the bullet power, and shoots the bullet to the tank-Intersection it’s facing to or tank-Output (the last section for the safety path). Then the tank-Conveyor will start to move back and forth to stand for the conveyor belt is moving.
Tank-Intersection	The tank is hit by tank-Conveyor from behind, which represents that it receive the package, and shoot to the tank-Conveyor its facing to that means the package is transmitted to the next segment.
Tank-Output	As long as it is hit, it expresses that the package in its safety path has already transmit from source to destination successfully. This tank then sends the message to teamleader to convince the fact that the mission of transmitting is complete.

The teamleader counts the total number of the mission-completion message from each safety path, when all path transmissions are properly accomplished, the teamleader then starts to decide the safety path logic for next round if there is any package waiting to transfer. When no packages are left in the conveyor system, the Robocode Simulation completes.

III.6 Java Animation double-buffering

No matter which tank shoots a bullet, it can save the location and time information to a text file using RobocodeFileWriter. As a simple but productive Java animation, the template code has several structures: the conveyor belt border and intersection drawing, settling down the corresponding threads on typical package size and speed definition. The Java animation also applies double buffering to eliminate the screen flash problem. The procedure for double animation manifests below:

1. Create an Image object *DbBuffer* by *createrImage(int width, int height)*.
2. Create an Graphics object *GraImage* by *DbBuffer.getGraphics()* in order to distribute and save the object needs to be paint in the memory space.
3. Use the repaint function *paint(GraImage)* to draw the canvas to the memory space.
4. Use function *drawImage(DbBuffer,0,0,null)* defined in *Graphics* to draw all of the animation window in one-time.

Chapter IV

Evaluation

IV.1 GReAT Transformation Time

With the help of GReAT, we transformed a domain-specific model defined in the conveyor experiment language to the target domain-specific model defined by grid meta-model. However, the transformation time is slow and in Table IV.1 presents the average time of transformation.

Table IV.1 Average Transformation time with deferent System Design

Conveyor Experiment Design	Average Time(mm:ss)
One Input, One Output, One Turnaround	00:1.21
One Input, One Output, Two Turnaround	00:3.56
Two Input, Two Output, Two Turnaround	00:13.60
Two Input, Two Output, Four Turnaround	02:25.63
Three Input, Three Output, Three Turnaround	01:49.30
Three Input, Three Output, Six Turnaround	12:59.08

According to the dimension of the conveyor system design, the transformation time increased exponentially. The logic design for creating connections among nodes should be refined. In current work, the GReAT spends 80% of its time to treat the connection parts in order to ensure the connection is correct.

IV.2 Drawbacks of Robocode

Before we start to evaluate Robocode, several general rules we use need to be explained (in Table IV.2) [24],

Table IV.2 Battle properties in Robocode

Rule No.	Description
1	When you shoot your gun you can select a power range from 0.1 units to 3 units. After firing you must let the gun cool down before firing again.
2	The heat generated by firing a gun is 1 + (energy amount/5).
3	Heat dissipates at the rate of 0.1 units per tick. For example, if you fire a gun with 2 units of energy the heat is 1 + (2/5) or 1.4 which means you cannot fire again for 14 ticks.
4	Robots start out with a fixed amount of energy (100 units) which is lost different ways. When the energy is gone the robot blows up.
5	Firing Gun = losing energy equal to bullet energy setting. If you set a bullet with 2 units of power the robot energy is reduced by 2.
6	Getting hit by bullet = 4 * bullet power + 2 * ceiling (bullet power – 1) For example, getting hit by a bullet with power 2 gives = 4 * 2 + 2 * (2-1) = 10 * 1 = 10
7	Shooting another robot gives you back some energy of 3 * bullet power. Suppose you fire with a bullet power of 2: Fire bullet = lose 2 units of energy Hit other robot = 3 * 2 = get back 6 units of energy

	Altogether increase of 4 units of energy
8	Max rate of rotation of robot: (10 - 0.75 * abs(velocity)) deg / turn. The faster you're moving, the slower you turn.
9	Max rate of rotation of gun: 20 deg / turn. This is added to the current rate of rotation of the robot.
10	Max rate of rotation of radar: 45 deg / turn. This is added to the current rate of rotation of the gun.

IV.2.1 Simulation Limitations

IV.2.1.1 Tank Life

According to the general rule above, we require that each tank's initial life energy is 100. There are three ways to affect the tank's energy that manifests in the Rule 1, 5, 6, 7. If a tank continuously hits another tank, then the shooting tank's energy will keep increasing by *2*bullet power*. However, meanwhile if the tank is continuously under attack, the total energy value will be reduced by *2*bullet + 2 ** ceiling **(bullet power - 1).* Namely the tank-Conveyor, tank-Intersection and tank-Output species of tanks will gradually lose their energy until the life energy is evacuated. So for CPS students' design, they cannot set an excessively large number of packages, since if the tank's life energy is zero, then the tank will vanish, which will destroy the original logic route.

IV.2.1.2 Heating and Cooling

Because of Rules 2 and 3, after the tank shoots, the gun barrel will generate heat to limit the frequency of tank firing. Once the heat value becomes zero, the tank can shoot again. In order to allow all bullets to emit and ensure all tanks on the safety path is well-prepared (no rotation of robot, gun and radar, according to Rule 8, 9, 10), we set a default waiting time by iteratively executing *doNothing()* for 40 time units to ensure sufficient time for the firing tanks and to confirm that the tanks' heading has been rotated to the anticipated direction. Hence in the real-time distributed Cyber Physical System world, this delay can cause errors. We need to consider how to eliminate those delays in the simulation.

Chapter V

Future work

V.1 Design Strategy

In Industry, the conveyor's transfer behavior is not simply transmitting a package from one input port to an output port. Rather, the package is usually sent from an input port to a Transfer server (A transfer server is a kind of transfer station, which is similar to output ports, but it is not the destination.), and then transfer to another transfer server, and so on. Finally the package will be transferred from one transfer server to the destination output port, and we should refine our design to meet this scenario requirement.

V.2 Simulation Improvement

V.2.1 Multiple packages

In our current Robocode, each safety path can only transfer one package. In the real-world, this kind of situation rarely happens. For instance, in our model design in Figure V.1, assuming that there is a current safety path transmitting a medium package from InputOne to OutputTwo, the safety path is InputOne->Intersection11->Intersection21 ->Intersection22->OutputTwo. Before transmitting the package, this route is busy, once the package passes Trunaround11 and turns to the second line, then Turnaround11 will be free, so if there is a small package needing to be transmitted from InputOne to OutputOne, then we should send the package without having to wait for the previous package to reach the destination. This can increase the veracity of our simulation.

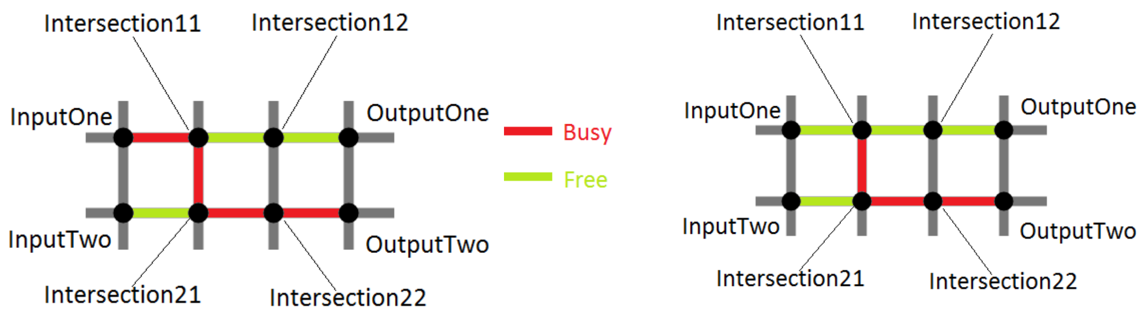

Figure V.1 (a) the safety path illustration from InputOne to OutputTwo

(b) The package transmits across the Intersection11 and

frees the first segment

One possible solution is better utilization of the tank's radar in Robocode. Instead of generating all of the logic using tank-Teamleader in the beginning of each round, the other kinds of tanks can dynamically report and scan the other tanks to know which part of the route is free in order to transmit the package without collision. Of course, because of the simulation limitations we discussed in Evaluation section, we may replace the Robocode simulation and apply a new approach to overcome those confinements.

V.2.2 Using a Physical Engine

The Robocode's physical simulation part is mainly defined by the bullet-shooting event; the different packages transmit with different velocity by the configuration of the bullet power [23]. In a real-world conveyor system, the physical environment is much more complex. Now in our Robocode we combined the cyber and the physical simulation, in future work, we would like to try to use some physical engine in order to help to establish the physical environment and simulate the behavior, for example, real belt speed, gravity, friction force, belt slop etc. The background simulation should be generating the cyber logic from Robocode first, and simulate the belt in the physical engine, the cyber and physical part connect with each other by instrumentation interface.

V.2.3 Refine the animation

Now the conveyor belt animation is simply a mapping result using the coordinate and time information, it can successfully present the route logic as well as feedback the transmit speed. In future work we would try to design a code based 3-Dimension simulation platform, to make the final feedback more like a conveyor belt system, so that meet the high standard and anticipation from CPS students.

V.2.4 Failure Tolerance

In our current safety path, no collision will happen, however, failure tolerance is one of the most important issues to be considered, such as if a transfer server shuts down or crashes, how to treat the situation to keep the whole line flowing.

V2.5 Scheduling Multiple Experiments to the Grid

As an online-based CPS-LaaS, another main area of this study is how to allocate space in the global grid for students to map their experiments to the grid and simulate in the background. In this case, figuring out the scheduling problem is important especially as there are limit resources in the grid.

The first approach plan is based on the assumption that the space of the global grid is big enough to handle all experiments. In order to calculate the efficiency of the scheduling methods, we first calculate the area utilization of the total nodes of grid in use. For instance, in Figure V.2, the red region and green region represent two different experiments, although the total area of those two experiments in two scenarios are same, the left one has lower space availability than the second graph. The algorithm needs more consideration to map as many experiments as possible.

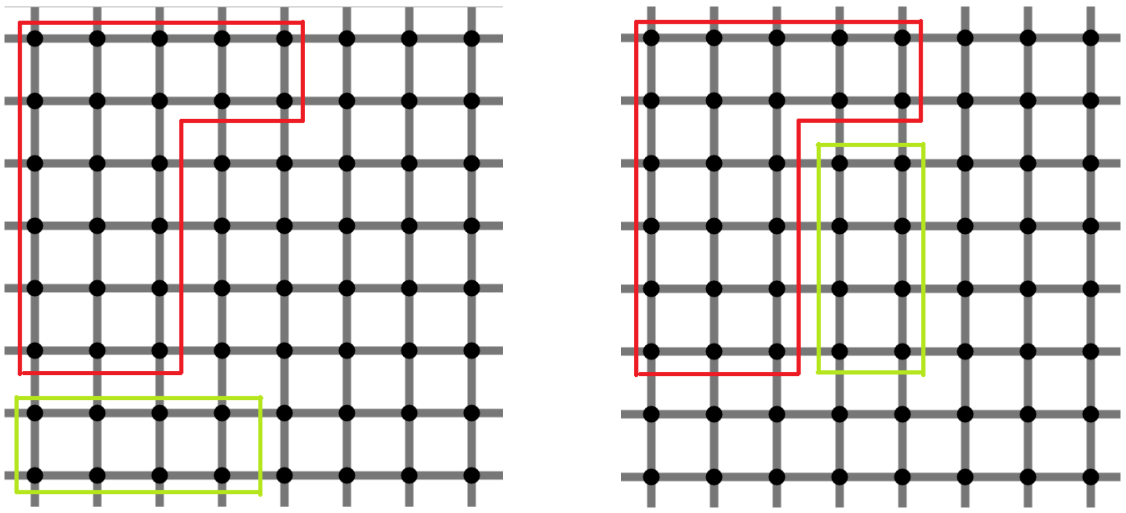

Figure V.2 Two scenarios of mapping two experiments

The second approach is focused on the spacing limitations of the global grid, so in some parts of the grid, the nodes need to be shared with multiple experiments, as Figure V.3 shows. In this case, we plan to use time sequence method to treat this situation. In one time period, the overlap area is conducted by a red experiment, and in next time period, the green experiment has the priority to use the grid, and then in the following time, the red region is using again, and so on, until both experiments have been completely simulated.

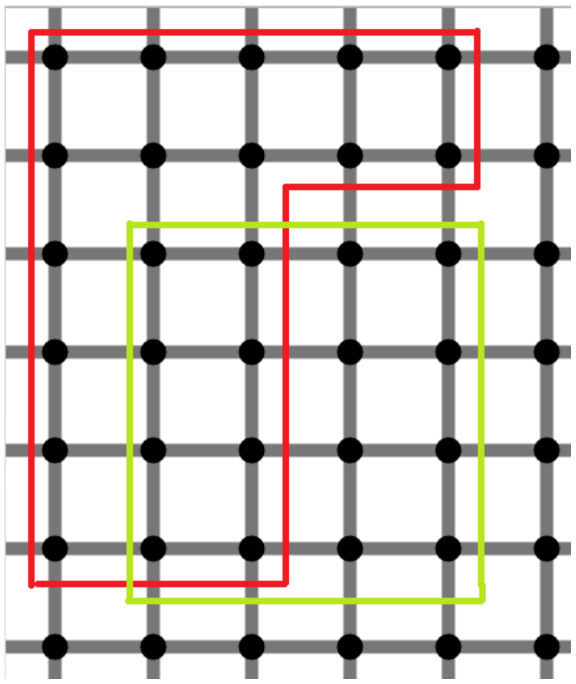

Figure V.3 Two experiments overlap and share the grid’s resource

Chapter VI

Conclusion

The motivation of this research is to develop a cost effective, easy to use and scalable online CPS laboratory. Owe to the convenience of the internet, we can handle a large number of requirements from CPS students to simulate their system design, for instance, a system design of reconfigurable conveyor system. In order to manage such an enormous number of experiments, we came up with an idea of the **global grid**, which is a large panel to allocate space to simulate the student's design.

In this thesis work, we focused on only one experiment and completed the whole process to simulate this experiment. There are two main parts; the first one is map the student-experiment design to the global grid, within this step, we make a reduction from mapping a complex system design with multiple components (e.g. **Input ports**, **output ports**, **turnaround**, **segment**) to a simple system design (namely the global grid, which only contains **nodes** and **connections** among nodes). Our complex system design and simple system design are all instances of the corresponding domain meta language defined in GME, and the transformation is made by GReAT. The second main part is simulation. We apply surrogate simulation software, Robocode, to simulate the behavior of a reconfigurable conveyor system. Since it is run in the background, the users cannot see how this software works. The final animation as the performance of the student's experiment design will be presented to the student's screen, which is generated by the information from Robocode. Now the whole process for one experiment has been accomplished, and our next urgent work is to treat the scenario of mapping and simulating multiple experiments on-line.

REFERENCES

[1] A. Gokhale, G. Biswas, N. Sarkar, S. Sastry, and M. Branicky, “CPS Laboratory-as-a-Service: Enabling Technology for Readily Accessible and Scalable CPS Education,” in Proceedings of the First Workshop on Cyber-Physical Systems Education (CPS-Ed) at CPSWeek 2013, Philadelphia, PA, USA: IEEE, Apr. 2013, pp. 21–24.

[2] K. An, A. Trewyn, A. Gokhale, and S. Sastry, “Model-driven Performance Analysis of Reconfigurable Conveyor Systems used in Material Handling Applications,” in Second IEEE/ACM International Conference on Cyber Physical Systems (ICCPS 2011). Chicago, IL, USA: IEEE, Apr. 2011, pp. 141–150.

[3] Dynamic Conveyor Corporation, “Reconfigurable Modular Conveyors: Equipment that Unites Controllers and Engineers,” Online Article in Medical Design Technology (MDT) Magazine, Jun. 2010.

[4] Yuan，L.，& Powell，S.(2013). MOOCs and open education: Implications for higher education [DB/OL].http://publications.cetis.ac.uk/2013/667.

[5] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica et al., “A View of Cloud Computing,” Communications of the ACM, vol. 53, no. 4, pp. 50–58, 2010.

[6] K. Ogata, System Dynamics, 4th ed. Prentice Hall, 2003.

[7] R. Rocha, L. Martins-Filho, and R. Machado, “A Methodology for the Teaching of Dynamical Systems Using Analogous Electronic Circuits,” International journal of electrical engineering education, vol. 43, no. 4, pp. 334–345, 2006.

[8] Lee, E A. Cyber physical systems: Design challenges [C] // Proc of ISORC. Piscataway, NJ: IEEE, 2008: 363-369

[9] John H, Marburger I, Floyd Kvamme, et al. Leadership under challenge: Information technology R&D in a competitive world. An assessment of the federal networking and information technology R&D program [R/OL]. Washington, DC: President’s Council of Advisors on Science and Technology, 2007. [2010-06-01]. http://qqq.nires.gov/pubs/

[10] NSF Workshop on Cyber-Physical Systems [EB/OL]. Austin, TX: Carnegie Mellon University, 2006 [2011-12-10]. http://varma.ece.cmu.edu/CPS/

[11] Kuang Z, Hu L, Zhang C. Research on Human Sensory Architecture for Cyber Physical Systems[J]. Journal of Networks, 2013, 8(9).

[12] J. Sztipanovits, and G. Karsai, “Model-Integrated Computing”, Computer, Apr. 1997, pp. 110-112

[13] Ledeczi A., Maroti M., Bakay A., Karsai G., Garrett J., Thomason IV C., Nordstrom G., Sprinkle J., Volgyesi P. The Generic Modeling Environment. Workshop on Intelligent Signal Processing, Budapest, Hungary, May 17, 2001.

[14] Agrawal, Aditya and Karsai, Gabor and Ledeczi, Akos, "An end-to-end domain-driven software development framework", OOPSLA '03, 2003, pp8--15, Anaheim, CA, USA.

[15] H. Su, G. Hemingway, K. Chen, T. Koo. Model-Based Tool- Chain Infrastructure for Automated Analysis of Embedded Systems. Fourth International Symposium on Automated Technology for Verification and Analysis (ATVA). LNCS, vol. 4218, pp. 523-537, Springer-Verlag, Beijing, China, Oct. 2006.

[16] Kleppe A,Warmer J,Bast W. MDA explained: The practice and promise of the model driven architecture [M].River: Addision-Wesley, 2003.

[17] Eisenstein J. Evolving robocode tank fighters[J]. 2003.

[18] Hartness K. Robocode: using games to teach artificial intelligence[J]. Journal of Computing Sciences in Colleges, 2004, 19(4): 287-291.

[19] Robocode-RoboWiKi http://robowiki.net/wiki/Robocode

[20] CTemplate. https://code.google.com/p/ctemplate/

[21] W. Wolf, "Cyber-Physical Systems," Computer, vol. 42, no. 3, pp. 88–89, 2009.

[22] Roberts C M. Radio frequency identification (RFID)[J]. Computers & Security, 2006, 25(1): 18-26.

[23] Floerkemeier C, Sarma S. RFIDSim—a physical and logical layer simulation engine for passive RFID[J]. Automation Science and Engineering, IEEE Transactions on, 2009, 6(1): 33-43.

[24] Rules and Features of Robocode Robots.
http://kss2.sd23.bc.ca/chalmers/cs11honors/robocode/roboPrograNotes/features.htm