|  |  |  |
| --- | --- | --- |
| (Graduate dissertation) | | |
|  |  |  |
|  | Fon52500 |  |



Document history

Revisions

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Status | Date | Changes |
| 0.1 | Draft | 14/12/2010 | Document Creation |
| 0.2 | First complete version | 10/01/2011 | First complete version |
| 0.3 | Reviewed version | 13/01/2011 | Problem description, conclusion, approach and Vanderlande chapters changed. |
| 1.0 | Finalized version | 14/01/2011 | Minor revisions |

Approvals

This document requires the following approvals

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date of approval | Name | Job title | Signature |
| 1 |  | Cees van Tilborg | Teacher |  |
| 1 | 14/01/2011 | Maarten de Koningh | Senior simulation software engineer |  |
| 1 | 14/01/2011 | Ivo van Wolferen | Senior simulation software engineer |  |
|  |  |  |  |  |

Distribution

This document was sent to:

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date of sending | Name | Job title |
| 0.2 | 10/01/2011 | Ivo van Wolferen  Maarten de Koningh | Senior simulation software engineer |
| 0.3 | 13/01/2011 | Ivo van Wolferen  Maarten de Koningh | Senior simulation software engineer |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Preface

Eighty -eight conveyor belts, of eight different kinds, spread over forty -eight zones along with a switch section and thirteen Vertibelts. That is just an example of the equipment in an emulation project at Vanderlande. What to do when all that equipment needs to be tested to make sure that there are no catastrophic failures when the customer flips on the power switch?

You emulate the equipment in a computer model.

That is what Vanderlande Industries is doing at the systems simulation department in Veghel. Testing with live equipment is often unfeasible and sometime impossible. The solution is to program the behaviour of the equipment into a computer model and fool the equipment’s controllers into thinking they are working with the actual hardware. This way the tests can be performed before the actual deployment of a system.

Currently emulation models are made with a modelling program called AutoMod and a framework built by Vanderlande in Visual Basic 6. But there is a need for a framework in a more modern language, Java, and with less AutoMod code, to decrease model lead-time and increase the model quality.

This document reports about the research and design to put together a new framework in Java. Chapter 4 introduces Vanderlande. The assignment is described in chapter 5. Chapter 6 gives a problem description. Chapter 7 details the approach. The research is discussed in chapter 8. Chapter 9 details the design. Chapter 10 describes the result. Chapter 11 is about the recommendations and risks involved and you will find the conclusion in chapter 12. Finally, you can find the evaluation in chapter 13.

**Table of Figures**

[Figure 1 Objects on conveyor belts and photo-eyes in AutoMod 8](#_Toc282768722)

[Figure 2 The buildings of Vanderlande in Veghel 9](#_Toc282768723)

[Figure 3 Communication diagram 11](#_Toc282768724)

[Figure 4 Internal communication comparison 16](#_Toc282768725)

[Figure 5 The main classes that make up the framework 20](#_Toc282768726)

[Figure 6 The matching process 21](#_Toc282768727)

[Figure 7 Event handling by the AutoModCommunicator 22](#_Toc282768728)

[Figure 8 Matching Interface 23](#_Toc282768729)

[Figure 9 Emulation Interface 24](#_Toc282768730)

[Figure 10 The test system in AutoMod 26](#_Toc282768731)

Contents

[1 Summary 5](#_Toc282768732)

[2 Glossary 6](#_Toc282768733)

[3 Introduction 7](#_Toc282768734)

[4 Vanderlande 9](#_Toc282768735)

[5 Assignment 10](#_Toc282768736)

[6 Problem description 11](#_Toc282768737)

[7 Approach 12](#_Toc282768738)

[8 Research – design considerations 13](#_Toc282768739)

[8.1 Criteria 13](#_Toc282768740)

[8.2 Candidate designs 13](#_Toc282768741)

[8.3 Chosen designs 18](#_Toc282768742)

[9 Design 20](#_Toc282768743)

[9.1 Framework 20](#_Toc282768744)

[9.2 User Interface 23](#_Toc282768745)

[9.3 Templates 24](#_Toc282768746)

[10 Results 26](#_Toc282768747)

[11 Risks and recommendations 27](#_Toc282768748)

[12 Conclusion 28](#_Toc282768749)

[13 Evaluation 29](#_Toc282768750)

[14 Appendices 30](#_Toc282768751)

# Summary

This graduation assignment took place at Vanderlande Industries in Veghel. Vanderlande is active in the baggage handling and other material handling systems.

In the Systems Simulation department models for low level control emulation are developed. Low level control emulation is used to test the software of low level controllers, like PLC’s, without connecting them to actual hardware. Instead, the hardware is emulated by a computer model. Vanderlande is currently using a framework based on an old programming language and a modelling program called AutoMod to develop the emulation models.

Vanderlande has requested a new framework based on a more modern programming language, Java. This framework is supposed to help shorten the model lead-time and increase the model quality. Not only was the new framework supposed to be build in a different programming language, Vanderlande was also interested in new and original ideas to overcome the challenges at hand. To that end, the first step after the initial problem orientation was to try and come up with various design considerations. These considerations would provide alternative solutions to encountered problems.

After the design considerations were made they were evaluated on the basis of a few criteria and the chosen design was integrated into a new framework.

This framework would be used in two different phases where each phase has a different user. The ‘matching phase’ consists of reading a description of the system that is to be modelled and is used by the Emulation engineer. The Emulation engineer is responsible for writing the code, or templates, that emulates the behaviour of the equipment described in the emulated system. During the matching phase the Emulation engineer makes sure that the described equipment will be emulated by the right template.

The other phase is the ‘emulation phase’. The PLC engineer uses the program in this phase to monitor and test the PLC. He is able to influence the emulated system using the program and uses AutoMod to view a visual representation of the system.

A third user of the framework, a software Architect, is responsible for the constant growth and improvement of the framework. With the growing number of templates that use the framework, it is guaranteed that the framework will constantly need to add new and improve upon existing functionality.

Early builds of the program have been well received and tests with a small system have been successful.

# Glossary

PLC

Programmable Logic Controller. An industrial computer used to automate electromechanical processes.

*LLC*

Low Level Control. The direct control of equipment.

*HLC*

High Level Control. The control of higher level processes such as routing, load balancing and flow control.

*PD0 document*

Project Definition file. Detailed description of the entire system in XML format.

*Mark code*

Abbreviation of the name of equipment.

*Function*

Numerical notation used to differentiate between similarly named, but functionally different equipment.

*Isc id*

Numerical notation used to assign unique identifiers to equipment in a system.

*Template*

Code that specifies the behavior of equipment

*Exception*

The exception to the rule (not an error thrown in programming languages such as Java)

*Photo-eye*

Light sensitive sensor used to detect the presence of objects at certain locations.

# Introduction

Low Level Control (LLC) emulation is emulating signals for a low level controller like a PLC. The controller’s outputs are read and used to simulate the behaviour of equipment. That behaviour generates the inputs that are send back to the controller.

This process is commonly used to test the software of low level controllers before they are connected to the actual equipment. The emulation model allows PLC engineers to monitor the behaviour of the controller and identify any bugs or errors. Other uses of the model include prototyping, i.e. proving that a concept can work, and validation, determining the capacity of a system and measuring its performance.

Vanderlande also performs High Level Control (HLC) emulation. This is a done in a Java framework and uses AutoMod to visualise the system. The HLC framework is used to test high level controllers and is not suitable for LLC emulation.

The main difference between HLC emulation and LLC emulation is perspective. HLC emulation models focus on the system as it is seen from the perspective of a high level controller. Examples of high level processes are routing, flow control and load balancing. LLC emulation models focus on the system from the PLC’s perspective. In this case it’s all about the details, the signals coming from the equipment.

In order to perform LLC emulation you need to know the exact behaviour of the equipment and how it is being controlled. The more accurate the behaviour of the simulated equipment is implemented, the more reliable the results of the emulation model will be.

At Vanderlande the equipment is described in a pd0 document. This document lists all the equipment in a system and all their attributes in a XML structure. This document is the basis for the LLC model development process. Equipment is identified by the *markcode* and *function* attributes, other attributes describe properties like the location of the equipment and equipment-specific parameters like the speed of a conveyor belt.

Although the structure of the document is the same for every project, the names of the equipment can vary between major projects and business units. E.g. a belt floor conveyor that is described by the *markcode* and *function* attributes *BF 02* in a project for Baggage can be different from a *BF 02* in a project for Distribution.

Emulation models consist of two parts. A logic part that controls equipment behaviour and a visual part that supplies a user interface and animates the model.

The behaviour of the equipment is defined in templates*.* A template consists of code, such as Java or Visual Basic and is written by the Emulation engineer. The templates describe what to do with input received from the controller and the output that should be sent back to the controller.

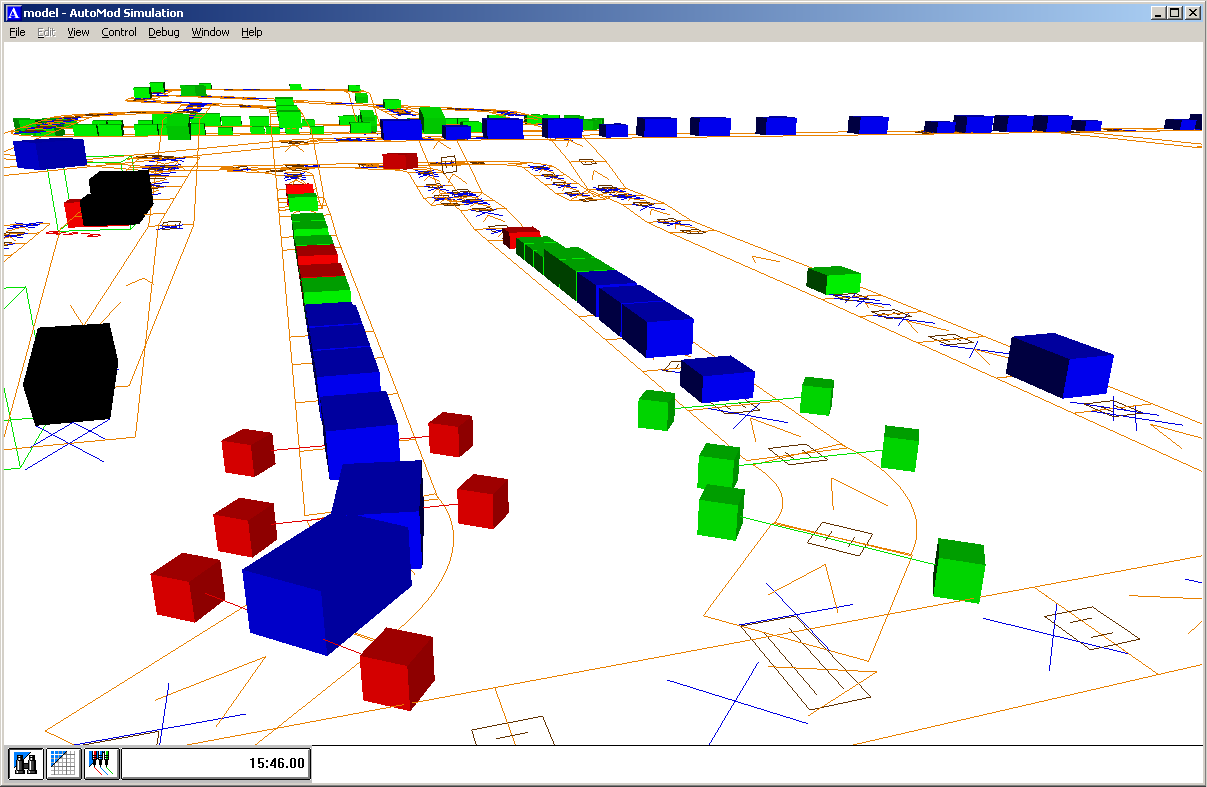


Figure Objects on conveyor belts and photo-eyes in AutoMod

The emulation models will be visualized by a program called AutoMod. This is a discrete event simulation software package that provides a 3d simulation of a system. AutoMod allows users to write code in an AutoMod-specific language to add functionality to the AutoMod model. For example, that code can be used to implement TCP/IP communication or to specify control logic for equipment. A screenshot of an AutoMod model is given in figure 1.

The LLC emulation framework has three users. The Software Architect, the PLC engineer and the Emulation Engineer. The PLC engineer uses the framework to test the PLC software. The Emulation engineer utilises the functionality available in the framework to develop templates. The templates use the framework to send and receive messages to the PLC and AutoMod. The templates also use the framework to allow the PLC engineer to interact with the system.

The software Architect is responsible for the constant growth and improvement of the framework. With the growing number of templates that use the framework, it is guaranteed that the framework will constantly need to add new and improve upon existing functionality.

# Vanderlande



Vanderlande Industries was founded in 1949 and is based primarily in Veghel. Vanderlande supplies automated material handling systems and accompanying services. The enterprise is active in the markets for baggage handling on airports as well as distribution centres and postal sorting facilities. There are about 2000 people working for Vanderlande with the majority based in Veghel but otherwise distributed across the globe. Vanderlande also has a software development team based in India.

Some well-known customers of Vanderlande are:

Baggage handling projects:

* Schiphol Airport
* Heathrow terminal 5
* Charles de Gaulle

Warehouse and Distribution projects:

* Albert Heijn
* Jumbo

Parcel and Postal projects:

* TNT
* DPD
* UPS

The assignment for this dissertation was performed for the Systems Simulation group. The Systems Simulation group of Vanderlande currently consists of 17 people, working in Veghel, Dortmund, and London. Their activities include:

* Simulations

These simulations can be used for prototyping and validation

* creating emulation models

These can be used to test the software of controllers and also for prototyping and validation

* Producing animations.

These are very useful demonstrating a concept or system to a (potential) customer.



Figure The buildings of Vanderlande in Veghel

# Assignment

Vanderlande currently uses a low level control (LLC) emulation framework written in Visual Basic 6.0 and AutoMod. This framework is used to test the PLC software that control various equipment like conveyor-belts, photo-eyes and barcode scanners.

Vanderlande wants to shorten the lead-time and improve the quality of emulation models with a new LLC emulation framework.

There are two reasons why Vanderlande wants a new framework instead of a revision of the existing framework:

1. Visual Basic 6.0 is old and its IDE is no longer supported.
2. The current framework depends too much on AutoMod which is a third party program.

The new framework should be written in a modern language and should be less dependent on AutoMod.

Vanderlande has chosen to use the Java programming language companywide for software development. This allows employees to keep working with the same language even when they change projects. Therefore the new framework will be written in Java.

The function of the LLC emulation framework can be divided into four areas:

1. Model development  
   The process of developing the models that can be used by the PLC engineers. This process begins with a document that specifies all equipment in a system. This process ends when all the relevant functionality in the system can be emulated. Several LLC emulation models are created every week. Having a short lead-time to make a new model is crucial.
2. Architecture and design of controls logic

The structure of the actual controls logic has to be flexible and easy to expand upon. New functionality is added frequently and should not require major overhauls of the framework.

1. Visualization using a physics engine  
   The emulation model has to be visualized and a physics engine has to handle the movement of objects including acceleration etc.
2. Manipulation of reality

When the PLC software is being tested it is necessary to introduce failures to the system and monitor the PLC’s behaviour. The PLC engineer needs some way of interaction to accomplish this. A graphical user interface is needed to accommodate for this.

Because the development of a complete framework takes too much time for the dissertation assignment, it has been decided to focus mainly on 1 or 2 aspects of the framework. Upgrading the visualization (3) is removed from the scope of this project entirely, instead the framework can use the current physics engine, AutoMod. The major focus of this assignment will be on the process of model development and the architecture and design of controls logic. The user interface is not a priority.

# Problem description

The problem domain can be described by a PLC that generates outputs and has to be sent correct inputs. We also have the AutoMod program that visualizes the system using a physics engine and that needs to be informed of the state of equipment when this impacts the animation. An example would be a conveyor belt that starts to move a piece of baggage. Last but not least we have templates that specify the behavior of equipment and require the outputs of the PLC and events from AutoMod.

The framework that needs to be built has to tie these three components together and make it work.

Figure Communication diagram

As you can see in figure 3, the framework is the center of LLC emulation. All communication has to pass through the framework. Every communication line in the figure has different information to send and receive:

Communication between the PLC and the framework consists of in- and outputs that change value.

Communication between AutoMod and the framework consist of events such as the arrival of baggage at certain locations.

The information received from both AutoMod and the PLC is sent to the templates that reply with the proper response if needed.

Before the communication has been established, there are two other jobs for the framework. First of all it has to facilitate the development of the templates. The framework should offer templates functionality designed to simplify and speed up the template development.

Second, the framework has to match the equipment in a system to the right templates. This is done by reading a pd0 document and identifying which equipment is emulated by which template.

# Approach

This assignment starts with a general orientation to examine the problem domain.

When the orientation was concluded a step by step plan was formed:

1. Determine global requirements
2. Research candidate designs
3. Choose candidate designs
4. Determine detailed requirements
5. Integrate chosen designs, implement and test

Step 1. Determine global requirements.

In this step the broad boundaries in which the new framework has to work are explored. This includes investigating the information found in the pd0 document, the behavior of the PLC and AutoMod and the nature of templates.

Step 2. Research candidate designs.

Now that the basic boundaries are know we can start experimenting with designs to overcome the challenges at hand. We try to approach the problems at different angles to come up with creative solutions.

Step 3. Choose candidate designs.

The candidate designs are weighed on a small number of criteria to determine which is best suited to use for the end product.

Step 4. Determine detailed requirements.

A continuation of step 1, we now try to specify exactly what Vanderlande wants the framework to do. The resulting requirements can be found in the appendices.

Step 5. Integrate chosen designs, implement and test.

With the detailed requirements and the chosen candidate designs we can start building the end product. When the product is build we will test it to see if it satisfies the detailed requirements.

# Research – design considerations

The research for this project consisted of creating various possible designs. These designs will be evaluated on a number of criteria, such as flexibility and complexity. One design will ultimately be chosen to be implemented.

The goal of these design considerations is to promote paradigm breaking. By looking at the framework from an outsider’s point of view it’s possible to find alternative, new ways to solve some of the challenges. In order to facilitate this process, this phase of the project will work only with a few loose criteria. The final requirements of the framework will be agreed upon when this phase concludes.

## Criteria

The candidate designs will be weighed on a trio of criteria.

1. Flexibility

The most important criteria is flexibility. Because there is no hard standard for the pd0 document, its interpretation is often different. The Emulation Engineers need flexibility to read and interpret the pd0 document. Because it’s impossible to predict the behavior of templates that will be developed in the future, the framework has to allow for a wide range of functionality in the templates. Emulation Engineers have to able to implement new functions on the fly without having to redesign the framework.

1. Performance

The next most important criteria is performance. The framework has to be able to process events in a real time manner. The PLC depends on the framework for information and the information has to be fresh. If, for example, a signal is delayed for only a second, the behavior of the PLC can be radically different.

1. Complexity

The last criteria is complexity. When the other two criteria are satisfied equally between candidate designs, the simpler design has the preference. This is because simple designs are often easier to implement and maintain.

## Candidate designs

The following problems have yielded multiple design solutions:

* Processing of the pd0 document
* Internal communication
* Communication with AutoMod
* Equipment initialization
* AutoMod functionality

### Processing of the pd0 document

The systems that the framework has to emulate are described by the pd0 document. This document is an xml file that details all the components of the system using a Vanderlande standard. In theory this standard allows engineers to know everything about the system from the pd0 alone. As it turns out, however, the standard is not always adhered to. For example, a belt floor conveyor with the mark code ‘BF’ and function variant ‘02’ should always be the same belt floor conveyor. Unfortunately this is not always the case. Therefore the framework has to account for some flexibility when processing the pd0 document so engineers can react to any deviations from the standard.

The structure of the pd0 document is also liable to change between projects. The framework has to able to identify the information that is relevant and be able to ignore that which is not. When the required information cannot be found, this has to be communicated to the user in a meaningful way.

There are essentially two ways to process the pd0. You can interpret the pd0 document and generate code that will have the desired functionality when it’s compiled. The alternative is to read the pd0 document and instantiate the objects that implement what’s needed. LLC emulation engineers have chosen for the former whilst HLC emulation engineers prefer the latter.

#### Generation

Currently the LLC emulation framework generates LLC models using a generator. Emulation model code is generated based on templates combined with the entries pd0 document. The model is made by compiling the generated code. The advantage of this approach is that the configurable part in the generator and the generated code can then be edited before it is compiled and used. This allows the engineers to change anything they want without having to change the PD0 document. It’s a very flexible setup, any exceptions in the matching process can be handled manually, keeping the generator relatively simple. The downside is that you need to compile the code before you are able to use the program.

The templates specify the functionality of the equipment and the generator extracts the necessary data from the pd0. This means that the data is defined in two separate places.

**Benefits:**

* Flexible

**Drawbacks:**

* Needs to be compiled
* Two points of definition

#### Instantiation

Meanwhile HLC emulation models are directly instantiated from the PD0 document. The HLC emulation framework does not need to be that flexible so they have chosen to instantiate rather than generate their code. This way they are able to avoid having to compile the program before using it and have a single point of definition. Because there is no manual intervention in the processing of the pd0, the instantiation process has to be able to handle every exception, which makes it complex and rigid.

Because of the reduced flexibility engineers have started to branch the source code for every (major) project. This has resulted in four different versions with challenging source code control.

**Benefits:**

* No compiler needed
* Single point of definition

**Drawbacks:**

* Less flexible

#### Hybrid

A new approach is to try and combine the best of both worlds. The framework will read the pd0 and provide the user with suggestions for all the equipment found therein. These suggestions are based off a list that couples equipment identifiers with classes and parameters. The user will then be able to edit the suggestions when necessary.

This is very useful when information in the pd0 is missing or incorrect. A belt floor conveyor that is specified in the pd0 without a defined speed can have its speed entered directly by the user. This eliminates the need to edit the pd0 document. The pd0 document cannot be changed because it is used in multiple departments and is already finalized.

Only when the user is satisfied with the classes and parameters will the framework instantiate the suggested classes with their suggested parameters. No compiler is involved and the user does not have to adjust the pd0 document.

This way you keep all the flexibility of generating without having to recompile or have a separate generator program. This could potentially be even more flexible because changes no longer require a compilation step and can be done with only a few clicks.

**Benefits:**

* Flexible
* No compiler needed
* Single point of definition

**Drawbacks:**

* Framework will be more complex and more difficult to implement

### Internal communication

Information received from AutoMod, the PLC or the GUI generally has to be handled by specific equipment. For example, AutoMod might notify the framework that a piece of baggage has reached the end of a conveyor belt. The framework has to reply with the next destination of the piece of baggage. That destination is known by the class that represents that conveyor belt. There are two ways to find that destination.

#### Observer pattern

One way to do so is to use the Observer design pattern. The Observer pattern is a topic-based publish/subscribe design pattern. Listeners subscribe to subjects when they want to receive updates about the subject. When the subject is updated it creates a message and sends it to all its listeners. The listeners inspect the message to see if the update applies to them and otherwise discard it.

In our example we would have one class in the framework that would broadcast the AutoMod message to all the equipment. Only the belt floor conveyor in question would reply to the message with the desired destination and the other equipment would ignore the message.

Although this implementation works fine in small numbers, it does not scale well. Every message will be sent to and evaluated by every equipment in the framework. Therefore the time required to process a message increases linearly for every equipment in the framework. This means that the performance of the framework would be negatively impacted when there is a lot of equipment.

*Design Patterns[1995]* by Gamma et al. states that the intent of the observer pattern is to “*Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically*.” However, the vast majority of the time, an event will only be handled by one listener.

Testing and debugging also becomes more difficult the more equipment there is. If you want to manually step through the process of event handling it will take a very long time when the event is sent to potentially hundreds of different template instances.

**Benefits**

* Works well with one-to-many relationship between events and listeners
* Implementation can be copied from current HLC emulation framework

**Drawbacks:**

* Does not scale well
* Debugging and testing can be time consuming

#### Content based publish/subscribe pattern

In the content based publish/subscribe pattern messages are intelligently routed to their final destination based on the content of the message. Subscribers have to inform the event handler of the messages that they want to receive. When an event is handled, the event handler identifies the appropriate subscriber and does not have to update the other subscribers.

In our example we would have one class in the framework that would inspect the AutoMod message and compare it to a list of subscriptions to identify the related equipment. Only the belt floor conveyor in question would receive the message and replies with the desired destination.

**Benefits:**

* Scales well
* Easy to test or debug
* Works well with one-to-one relationship between events and listeners

**Drawbacks:**

* Framework will be more complex and more difficult to implement

Figure Internal communication comparison

Both publish/subscribe patterns have been tested for performance, the results can be seen above in Figure 4. The test was conducted with 2 listeners that wanted to receive all events, 5 listeners that wanted multiple events and an increasing number of listeners that only respond to 1 event. 100000 events were generated and sent to the event handler. The content based event handler than extracted the key from the event using string manipulation and notified the appropriate listeners. The Observer based event handler notifies all its listeners of the event, who then proceed to evaluate it.

The Observer based pattern was expected to be faster initially but take linearly more time with more listeners. The results, however, show that the content based approach is practically always faster.

### Communication with AutoMod

AutoMod is the program used to visualize the emulation. It draws all the components and is also responsible for the transportation of items such as baggage. AutoMod currently supports two communication protocols. ActiveX and TCP/IP. It is not currently known which of the two protocols has the performance advantage, but both have been used successfully in emulation projects.

Communication between jNAS and AutoMod needs to be asynchronous. jNAS can send user-generated events to AutoMod at all times, and AutoMod cannot block because that would pause the entire model. If an event-only AutoMod model is used, AutoMod will frequently ask jNAS for instructions. When this happens and synchronous communication is used the model would freeze until jNAS responds. Because the PLC is very time-sensitive this behavior is undesired. Therefore it is important to utilize a communication protocol that supports asynchronous communication.

#### ActiveX

ActiveX is a Microsoft technology and only works in Windows. Using a COM object it is possible to call methods, read and set properties and react to events. This allows both synchronous and asynchronous communication. Apart from merely sending information, ActiveX can also be used to pause or unpause the AutoMod model.

**Benefits**

* A partial implementation can be copied from HLC emulation framework

**Drawbacks**

* Only works in Windows

#### TCP/IP

TCP/IP uses sockets to send data to an IP address. The AutoMod program must have some code that reads its socket and handles any received data. AutoMod must have a model running to be able to execute that code so TCP/IP cannot be used to start a model. A partial workaround for that is to use a command-line parameter when starting AutoMod that immediately starts the model. Because data is sent to an IP address it is easy to use when a model is run on multiple computers.

**Benefits**

* Implementation can be copied from current HLC emulation framework

**Drawbacks**

* Cannot be used to unpause a model

### Equipment initialization

The nature of the communication with the PLC requires the framework to hold a list with all the inputs and outputs. These in- and outputs are also used by the equipment classes. It is therefore arguable to have the equipment classes use the variables that are stored in the PLC communicator class. Instead of having every equipment instance hold its own variables.

#### Static equipment

When using static equipment, the equipment classes retrieve their variables from a central I/O list. This means that all data is defined only once and in one place. However, there are some variables that do not appear in the I/O list and would have to be stored somewhere else. Furthermore, the data retrieval may have a negative impact on event handling speed.

**Benefits**

* Single point of definition for most of the data

**Drawbacks**

* Retrieval of data may slow down event handling slightly
* Framework will be more complex and more difficult to implement

#### Instanced equipment

Having the equipment instanced allows for a 1:1 relationship with objects in the pd0 document. Every instance can hold its related variables and does not have to look these up in a list. This does mean that these variables now exist in multiple locations. Once in the I/O list and once in every class that has a relation to it. That introduces new problem in the form of synchronization. When a variable is updated, that update has to be pushed to all its instances. If the update happens when the variable is being used, it might lead to inconsistency.

**Benefits**

* Easier to implement

**Drawbacks**

* Synchronization can be an issue

### AutoMod Functionality

AutoMod is the program used to visualize the emulation. It draws all the components and is also responsible for the transportation of items such as baggage. But it is also possible to have AutoMod do more. AutoMod allows engineers to program functions in a C-like language. The current LLC emulation framework uses AutoMod for almost all the model logic. Examples of the AutoMod functionality are accelerating and decelerating the cargo, routing the cargo and triggering events when an optical sensor is blocked.

#### ‘Smart’ AutoMod

Having a smart AutoMod model means that AutoMod implements a lot of functionality. Having all the information in the AutoMod model means that events can be handled without having to communicate with an external program. AutoMod does not provide a user interface however, so an external program will remain necessary to allow the user to interact with the model.

**Benefits**

* Quick event handling

**Drawbacks**

* Tight coupling with AutoMod, very hard to switch to a different animation program

#### Event-only AutoMod

Having an event-only AutoMod model means that AutoMod handles as little functionality as possible. The AutoMod model is only responsible for the animation of the components. Whenever an event occurs, e.g. a piece of cargo reaching the end of a conveyor, the AutoMod model will relay that event to the jNAS framework to find out what the appropriate action is.

The main reason for this is to make the framework less dependent on AutoMod. AutoMod is an externally developed and owned program. It is unacceptable for models to cease to function because of an AutoMod patch or license discontinuation. With all the logic in the jNAS framework it is far easier to switch animation programs. It also makes it possible for jNAS itself to animate the model in the future.

**Benefits**

* Loose coupling with animation program to allow for other animation programs

**Drawbacks**

* Information needs to be communicated from AutoMod

## Chosen designs

### Processing of the pd0 document

The Hybrid design has all the benefits of the other two solutions with the only drawback of increased complexity. Because it does not need a compiler it is faster than the generation design. Because performance is more important than complexity it is the best choice.

### Internal communication

The Content based publish/subscribe design is the better choice for the same reasons. It has better performance than the observer pattern design although it is more complex. Performance is more important than complexity so the internal communication will be implemented using content based publish/subscribe.

### Communication with AutoMod

The TCP/IP design is easier to extend towards a distributed model. Since performance and flexibility are equal, this makes it the winning design.

### Equipment initialization

The Instanced equipment design is equally flexible, has better performance and is less complex. This makes it clearly the better choice.

### AutoMod Functionality

The Event-only design is more flexible than the ‘smart’ AutoMod design. Furthermore, the initial goal of the project was to minimize the reliance on AutoMod for LLC emulation. Therefore the Event-only AutoMod design will be implemented.

# Design

The design can be split up into three parts. The Framework is responsible for the core functions of the program. It deals with the matching process, communication with AutoMod and communication with the PLC.

Another part is the User Interface. The User Interface is responsible for showing relevant data to the user and allowing the user to interact with the program.

The third part is the templates part. The templates part is responsible for the emulation of equipment. Specifically the functionality that the Emulation Engineer can use to interact with AutoMod, the PLC, other equipment and the user interface.

## Framework

The Framework is responsible for the core functions of the program. These include the matching of elements in the pd0 document to templates, instantiating the templates, communicating with AutoMod and communicating with the PLC.

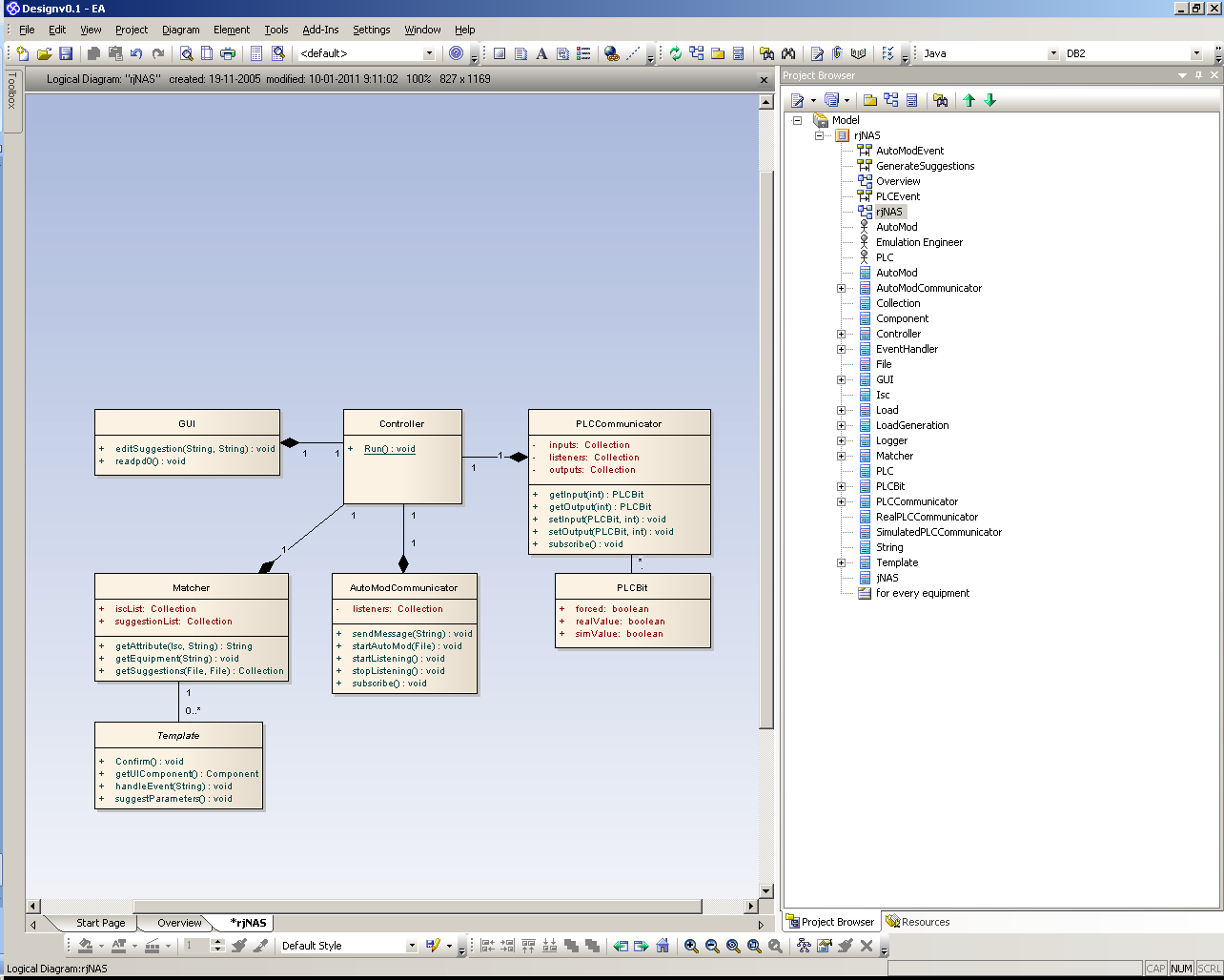


Figure The main classes that make up the framework

When the program is run, the Controller reads a properties file that describes the user and the default values for the input files. The input files are the pd0 document, the matching rules file and the AutoMod model executable.

The user can be one of two things. A PLC Engineer or an Emulation Engineer. If the user is a PLC Engineer then the Controller will skip the Matching Phase and proceed directly to the Emulation phase, unless there is an error in the matching process.

### Matching

The matching process reads the matching rules file and the pd0 document and uses the matching rules to provide the Emulation Engineer with template suggestions for the equipment.

The matching rules has three different kinds of rules; Exceptions, Parental Exceptions and Matches. The Exception rules match the equipment in the pd0 by their Isc Id parameter. The Parental Exceptions match the equipment by their Isc Id parameter and the Isc Id parameter of their parent equipment. That Matches rules match the equipment by their mark code and function parameter.

Here is a small example of a matching rules file:

EXCEPTIONS

com.vanderlande.sim.lolecoja.template.bf.BFspecial 1.1.3, 1.2.3, 2.1.7

PARENTALEXCEPTIONS

com.vanderlande.sim.lolecoja.template.bf.BF80 MFZ01&BF80

com.vanderlande.sim.lolecoja.template.bf.BF81 MFZ02&BF80

MATCHES

com.vanderlande.sim.lolecoja.template.bf.BF02 BF02, BF03, BF08

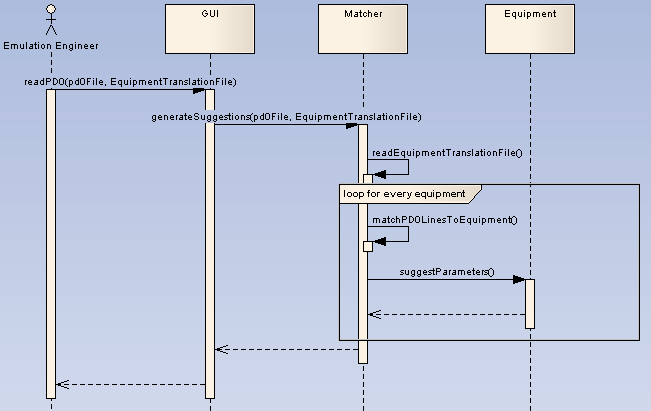
com.vanderlande.sim.lolecoja.template.TRZ TRZ01

com.vanderlande.sim.lolecoja.template.SVZ SVZ\*

As you can see, all the rules start with the name of a template class but have a different key at the end, depending on the kind of rule.

The Matcher class will read the matching rules file and store all the rules in three Maps, one for each kind of rule. These maps have key and value pairs. The template class will be entered as the value, and the matching parameter as the key.

When a rule is read with a template that cannot be found in the program, the user will be notified with a warning.

 Figure 6 The matching process

After the matching rules file is read, the Matcher class can read a pd0 file and start matching the equipment. The pd0 file is read using JAXB. JAXB is used to marshal an example pd0 file at development time and uses the structure of that example file to unmarshal pd0 files at runtime. This allows developers to work with the information in the pd0 file as if it were a Java class.

The Matcher will extract the relevant parameters of the equipment and compare these to the keys that are in the Maps. When a match is found the appropriate template will be asked to extract any information it needs from the pd0 document.

If a match cannot be made, or if the template cannot extract enough information, the user will be notified with a warning.

### Instantiating

When all the matches have been confirmed by the user through the User Interface all the matched templates are instantiated. The templates will subscribe to AutoMod and PLC events and initialize any User Interface components if needed.

### Communication

The templates need to be able to communicate with the PLC and AutoMod. Communication with the PLC is handled by the PLCCommunicator class and communication with AutoMod is handled by the AutoModCommunicator class. Both of these classes implement the content based publish/subscribe pattern. Templates can subscribe to these classes when needed.

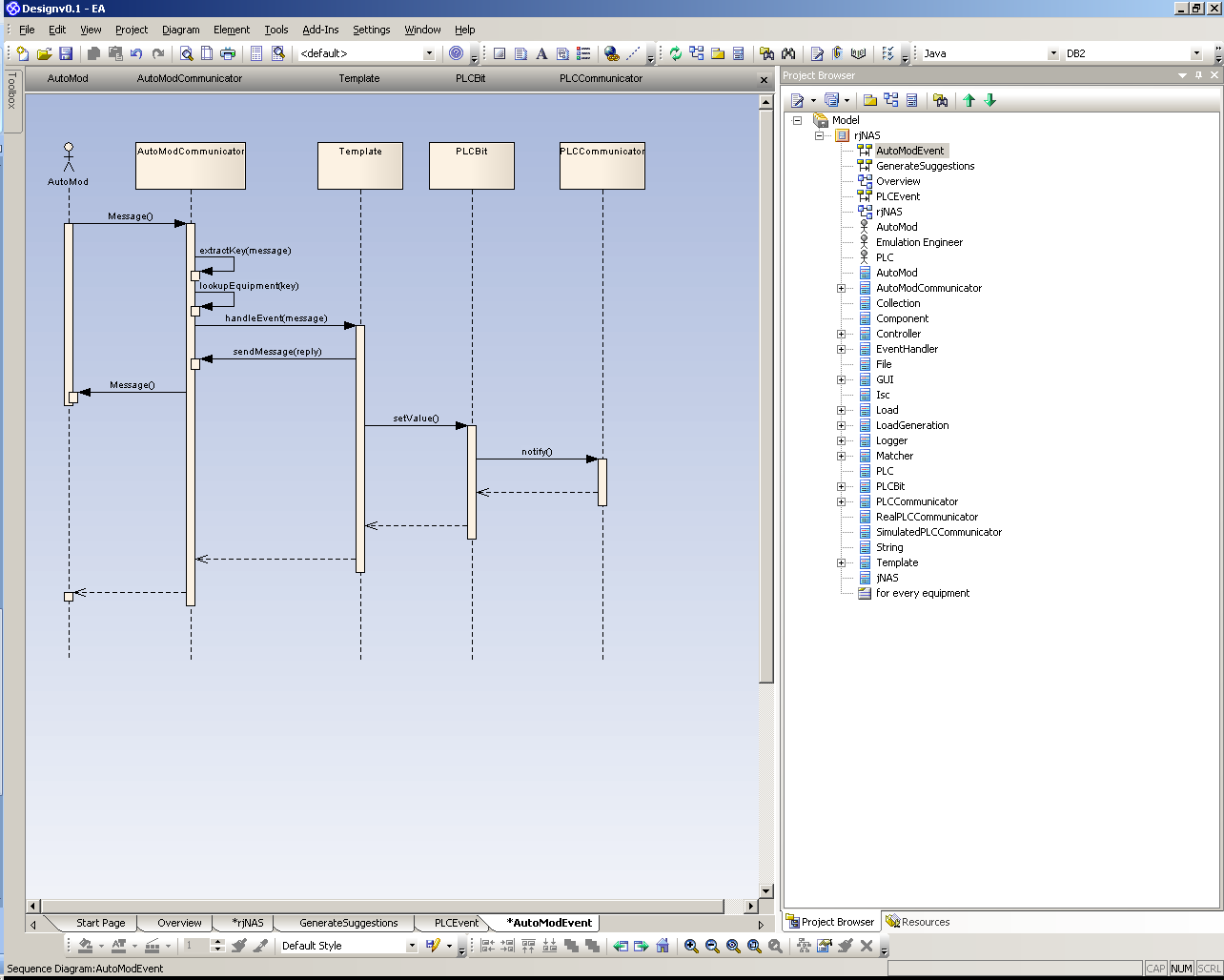


Figure 7 Event handling by the AutoModCommunicator

The process for event handling by the PLCCommunicator is identical with the exception of the key extraction and equipment lookup.

For the PLCCommunicator the bit address is used as content. Templates can subscribe to in- and output bits of the PLC based on the bit addresses. When a bit is changed, whether by the PLC or by the User Interface, the subscribing templates will be notified with the new value. This allows the templates to react by sending a message to AutoMod or the PLC, or by simply updating some internal values like a counter or flag.

If the value of a bit is being displayed by the User Interface, a message will be sent to the User Interface that notifies it of a change so it can update its values.

In the case of the AutoModCommunicator the content is the content of messages received by AutoMod. AutoMod sends and receives plain string messages. These messages include an event name which is used as the key to subscribe to. An example of such a key is the name of a conveyor belt. When AutoMod detects that a piece of baggage is at the end of a conveyor it will send a message with the name of that conveyor as the event name.

## User Interface

There are two different user interfaces, one for the matching phase and one for the emulation phase. The matching interface is used by the Emulation engineer to check and edit the template matches. The emulation interface is used by the PLC engineer to test and monitor the PLC.

### Matching phase

During the matching phase the user has to be able to specify the input files, i.e. the pd0 file, the matching rules file and the AutoMod model executable. This is done by simple text fields and browse buttons. The user also needs to be able to start the match generation, edit the matching rules, edit the matched templates and approve the matches.

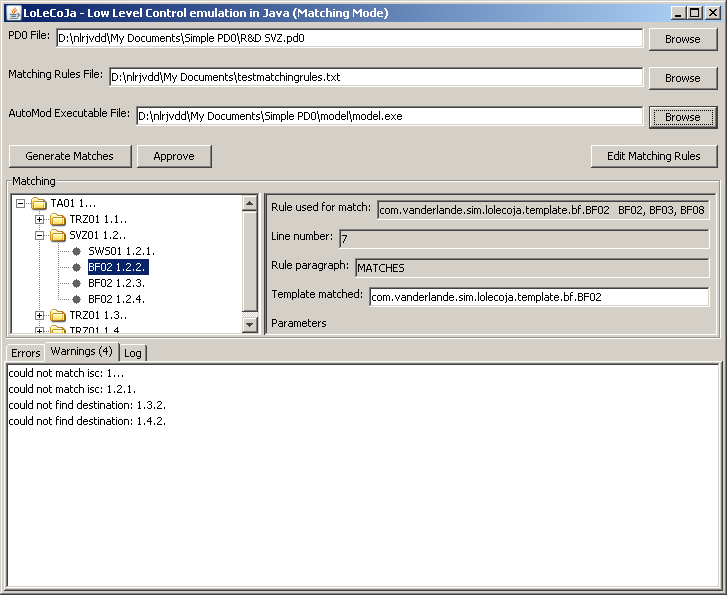


Figure Matching Interface

At the top of the screen are the buttons and text fields for the input files. In the middle of the screen are the equipment described in the pd0 document on the left side and on the right side is the matched template along with some information about the rule used to make the match. At the bottom of the screen are the Errors, Warnings and Log panel. The user can click on the Errors and Warnings lines and the program will jump to the equipment that caused the error or warning.

To start the match generation the user only has to click the ‘Generate Matches’ button. If the entered pd0 file and matching rules file are existing files the generation will begin. Otherwise the user will be shown an error in the bottom of the screen.

If the user wants to edit the matching rules file, he or she can click on the ‘Edit Matching Rules’ button and the matching rules file will be opened by the default program as set in the operating system. Generally this will be a text editor like Notepad or UltraEdit.

If the user wants to edit a matched template, he or she can type directly into the text field that shows the Template matched. A validator class will check if the entered text is a valid class and will change the background colour to green if it is and red if not. When the user presses Enter a popup will ask the user if he or she wants to add an exception to the matching rules file or not. The matched template will be updated to the new template.

When the user wants to approve all the matches, and there are no errors, the user can click on the Approve button and the program will close the matching interface and open the emulation interface.

### Emulation phase

During the emulation phase the user, which is the PLC engineer, has to be able to select equipment, view and edit parameters and interact with template-specific interfaces.

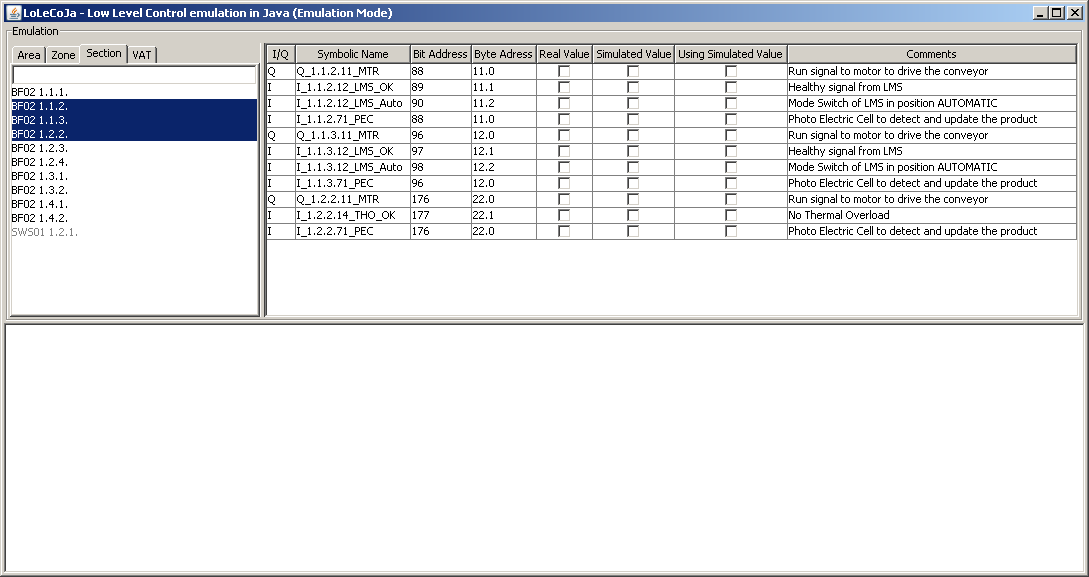


Figure Emulation Interface

In the top left corner the user can select equipment. The equipment is shown in a list by their mark code, function and Isc id. The user can filter the equipment by hierarchical level using the Area, Zone and Section tabs. The user can also filter by typing into the text field which will filter the equipment based on their mark code, function and Isc id. The user can select multiple equipment.

In the top right corner the user can see all the PLC in- and outputs that are related to the selected equipment. If the equipment has a specific interface it will appear below the in- and outputs.

The bottom portion of the screen is reserved for logging.

## Templates

Templates are the classes that are responsible for the actual emulation. The behavior of the conveyor belts, photo eyes and other equipment is defined in the template that they are matched to. The job of the templates is to provide the PLC with the proper inputs, based on the PLC’s outputs. For example, take a conveyor belt with a piece of baggage at the start and a photo eye at the end. When the PLC sets the output for the conveyor’s motor, the baggage should start moving towards the end of the conveyor and finally the photo eye should toggle its input to let the PLC know that the piece of baggage has reached the end of the conveyor belt.

### Matching

The Mainframe will match templates to equipment described in the pd0 document. When a match has been made, the template will be able to extract information from the pd0 document. For example, a template that simulates a conveyor belt needs to know the name of the belt, the destination of the belt, the PLC output that drives the motor of the conveyor belt and it needs to know the speed of the conveyor belt. The template has access to the unmarshalled Java class of the matching equipment. Using that class the template can read all the parameters of the matching equipment and its children. This should be enough for the template to initialize all its variables. If this is not the case the template has to log an error or warning.

The template will also provide the user interface with the names and values of all its variables. This allows the Emulation engineer to check for any unexpected values.

### Event Handling

In order for the templates to react to signals from the PLC or AutoMod, it needs to be notified of all messages that are relevant. For this reason the template will subscribe to the PLCCommunicator and AutoModCommunicator. When the template is notified of a message, the template will be able to react by sending a message to AutoMod or the PLC or do some internal processing.

### User Interface

To accommodate for templates that have require more user interaction besides the manipulation of the related PLC in- and outputs the user interface enables the templates to provide their own user interface component. These components can be used to show internal variables and allow more control for the PLC engineer.

An example of a template that needs more interaction is a VertiSorter. A VertiSorter can elevate a conveyor belt to switch between two destinations. The PLC engineer needs to be able to edit the time it takes for the elevating conveyor belt to switch between destinations. This way the engineer can see how the PLC responds to abnormal behavior. The template of the VertiSorter should provide a user interface component that allows the PLC engineer to edit the switching time.

# Results

The result of this assignment is an end product with innovative features such as content based publish subscribe and hybrid pd0 processing.

The implemented framework is able to:

* Read pd0 documents.
* Match the right equipment to the right template.
* Provide the user with clear errors when this is not possible.
* Communicate with AutoMod and the PLC.
* Provide templates with functionality to allow for easy communication.
* Provide templates with functionality to allow for easy and efficient event handling.
* Automatically expose the related in- and outputs of a template to the PLC engineer.
* Provide templates with the potential to display custom user interface components.

The code has been reviewed and approved in two review sessions by the Java expert of the Systems Simulation department.

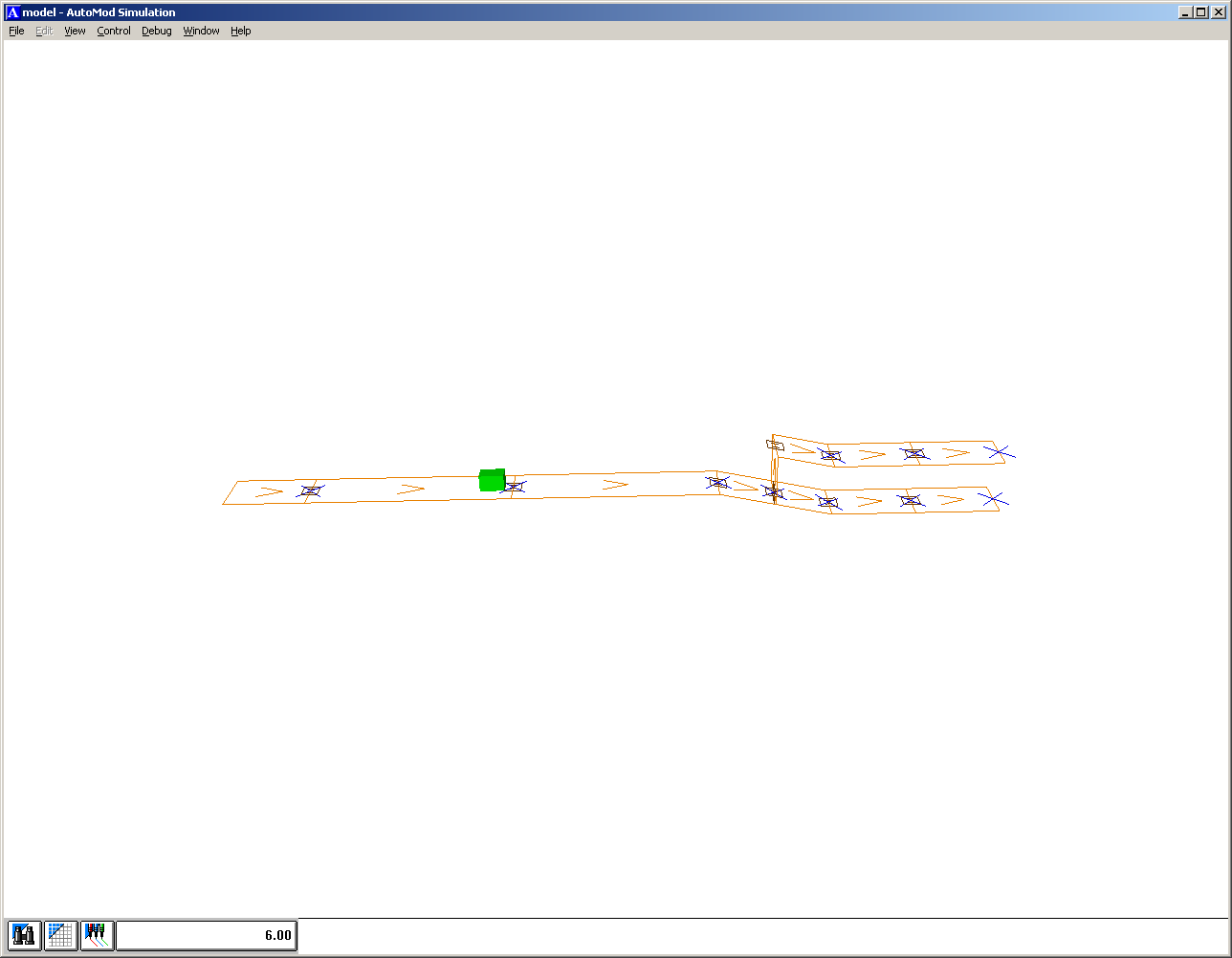


Figure The test system in AutoMod

Figure 10 shows the system that was used to test the framework. It consists of 10 conveyors, one of which that can be elevated or lowered to change destination. Using the framework it is possible to send packages to either the top conveyors or the bottom conveyors, depending on the position sensors of the elevating conveyor.

Looking back on the goals outlined in the assignment, I can say that the project has been successful in solving the problems of model development, architecture and design of controls logic and even a (incomplete) GUI has been implemented.

# Risks and recommendations

If the program is ever expanded I would like to see AutoMod removed from the framework and have the program animate the system by itself. This might make the creation of templates simpler. It would also allow the program to work at faster than real time speeds. The system would likely also be even more responsive if there is no need to send messages through TCP/IP.

Furthermore this might make it easier to completely automate the PLC testing. Unit tests could be added to the program that tests the behavior of the PLC. This would be something for further down the road though as such an addition would be quite complex to code.

The risks in this project are the AutoMod code and the unmarshalling of the pd0 document.

The AutoMod code needs to be able to handle all the messages that it receives and it needs to remain synched with the Java program. Although all the current LLC emulation has been done with AutoMod, this does not guarantee that all the functionality can be done the same way with this new framework.

The risk can be minimized by keeping the messages sent to AutoMod as simple as possible. Ideally the messages should only need to send information about the transportation of baggage pieces.

The risk with the unmarshalling of the pd0 document lies in the unreliability of the pd0 document. If the structure of the pd0 document changes the matching process can fail in two places. The unmarshalling itself can fail, or the matching of the variables can fail because the template is no longer looking in the right place.

The unmarshalling risk can be minimized by using the latest JAXB that supports the any-attribute. This allows the unmarshalling to parse unrecognized attributes. This is a good solution when the unrecognized attribute is not relevant to the emulation process, but otherwise this is not enough. For example, if the motor address of a conveyor belt is changed from a Symbol to something else, the template will not be able to match its address.

# Conclusion

To conclude this project, the results will be compared to the objectives described in the assignment:

1. Model development  
   The process of developing the models that can be used by the PLC engineers. This process begins with a document that specifies all equipment in a system. This process ends when all the relevant functionality in the system can be emulated. Several LLC emulation models are created every week. Having a short lead-time to make a new model is crucial.
2. Architecture and design of controls logic

The structure of the actual controls logic has to be flexible and easy to expand upon. New functionality is added frequently and should not require major overhauls of the framework.

1. Visualization using a physics engine  
   The emulation model has to be visualized and a physics engine has to handle the movement of objects including acceleration etc.
2. Manipulation of reality

When the PLC software is being tested it is necessary to introduce failures to the system and monitor the PLC’s behaviour. The PLC engineer needs some way of interaction to accomplish this. A graphical user interface is needed to accommodate for this.

**Model development.**

Both the hybrid pd0 processing design and the simple and efficient event handling both contribute to the decrease of model lead-time. There is no longer a compilation step in the process of emulation model development. When writing templates, engineers can now easily and efficiently handle events generated by the PLC and AutoMod.

**Architecture and design of controls logic.**

With the introduction to Java, engineers can now use object oriented programming and inheritance when programming templates. Exposing the related in- and outputs of a template to the PLC engineer is now handled automatically.

**Visualization using a physics engine**

As mentioned in the discussion of the assignment, this has been left out of the scope of this project.

**Manipulation of reality**

Besides the related in- and outputs, templates can also specify custom UI components. This is sufficient to allow the PLC engineer to manipulate all the desired parts of the system.

The implemented application has been well received and tests with a simple pd0 file have been successful. However, it was not possible to test all existing templates at this stage.

# Evaluation

Looking back at my graduation assignment I can say that I’m happy with the designs and implementation I have made. I am proud of my end product and think that it could be used for low level control emulation here at Vanderlande.

I have worked in a nice environment with helpful colleagues and had a great time. The project was quite a challenge and I learned a lot in the process.

The approach of researching design considerations to improve paradigm breaking has been successful and interesting. I have been able to come up with solutions that I might not have thought of otherwise. An example would be the hybrid processing of the pd0 document. The HLC emulation team currently uses instantiation and the LLC emulation team uses generation. Normally I might have picked one of those two, seeing as that they have already proven themselves in practice and it would be easy to adopt. Now, however, I have something that I came up with on my own.

Inspired by the results of my research, Vanderlande has also begun investigating alternatives to the observer pattern to use in the current HLC emulation framework. The hybrid processing of the pd0 document has also received positive feedback and is something that will definitely be used when Vanderlande implements their new LLC emulation framework.

I wish I could say that I was happy with the entire project but I suppose I can’t have it all. Writing this document was slower and more difficult than any document I’ve ever written. Even though I started with documentation on time, I did not progress fast enough and it came down to the last minute to patch things up.

I learned a lot about programming with Java which I’m sure is going to be useful in future jobs. I tried to choose the fastest classes and methods wherever I could and learned a lot about Java classes like hashmaps, treemaps, lists and arrays in the process.

I hope my program is used in some way after I am done with my assignment. Even if none of my code is used I am happy to have given my mentors at Vanderlande some new ideas.

# Appendices