Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Control Code Generation using Model Engineering for an Electric Train

Florent Frizon de Lamotte * **Pascal Berruet** * **André Rossi** *
Jean-Luc Philippe *

* *CNRS, FRE 3167 – Lab-STICC*
Université Europenne de Bretagne – UBS
Centre de Recherche, F-56321 Lorient Cedex – FRANCE

Abstract: This paper addresses first experiments in the usage of model engineering to generate control/command code from a high-level description of the system featuring the notions of architecture and configuration. This principle has been applied generate IEC-61131-3 control/command code for an industrial PLC to control the circuit and the engines of an electric train. It has been implemented using the AMMA platform, more precisely the ATL transformation language.

Keywords: Model-driven systems engineering; Flexible and reconfigurable manufacturing systems; Dependable manufacturing systems control

1. INTRODUCTION

Reconfigurable systems have gained much attention in the two last decade because of their ability to evolve to follow quality of service requirements. This article deals with the control/command code generation for reconfigurable manufacturing systems (RMS) [Merhabi et al., 2000] using model engineering. In fact, the control/command code for a reconfigurable system must take into account its reconfiguration capabilities and should offer reconfiguration primitives to the supervision level. This article focusses on the code generation for the machine level (sensors/actuators) and the cell level (coordination between machines).

Model engineering is a promising development from software engineering. In model engineering, everything is a model [Bézivin, 2005], which means that the code itself is a model until it is serialized into a textual form for implementation purposes. Models are described using a meta-model which in turn is a model. Transformations can be applied on models by defining mapping on their meta-models. We previously used this technology to perform the analysis of the architecture [Lamotte et al., 2007] or the configuration [Lamotte et al., 2006] of a reconfigurable system and it proved to be very efficient. These analyses needed to be performed for a real system. The DeSyRe [Lamotte et al., 2007] language designed for the description of such system and which have been used in previous work seemed to be suited as en entry point for the generation of the control/command code to run one such a system.

Model Engineering has already been used to perform code generation for control/command application such as in [Vogel-Heuser et al., 2005] or [Thramboulidis et al., 2004] where UML is used to describe the application. In most of these cases the program is itself written in UML. Here, the starting point of the code generation is a modelling of the process itself, in term of resources.

To experiment with this code generation, a demonstrator has been set-up. It consists in an electric train circuit described in section 2. The DeSyRe language, which is the input of the code generation is then described in section 3. Finally, section 5 describes the code generation process.

2. DESCRIPTION OF THE APPLICATION

The target application is an electric train circuit. Electric trains give the opportunity to work on large systems built upon small component that can be easily found. These systems are easy to set-up and to modify, so lots of case studies can be realized. Moreover reconfiguration is easily featured on such system, it consists in changing the path followed by the trains.

The circuit, studied in this article is presented on figure 1, it contains five turnouts and ten sections that make up three loops. The *inner loop* is constituted by the `T10`, `T21` and `T11` elements. Inner loop communicates with the *small loop* through the `A5` turnout, this second loop is composed of the `T4`, `T5` and `T1` sections. A third loop, which we call *outer loop* share `T1` with the small loop. The two other sections of the outer loop : `T19` and `T3` have been separated as they formed a unique but very long section. A `Fx` label has been placed on some sections, it denotes the capacity of this section to implement an operation realizing the `Fx` function.

The studied circuit features good flexibilities that can advantageously be used for reconfiguration. A criticality analysis performed according to [Lamotte et al., 2007] shows that 16% of the resources (sections and turnouts) are critical to the functioning of the system. In fact, by removing `T4`, it is still possible from section to perform any function.

Engines are from Marklin and use the digital system which enables the control by transmission currents. As described on figure 2, a Schneider-Electric/Telemecanique PLC is

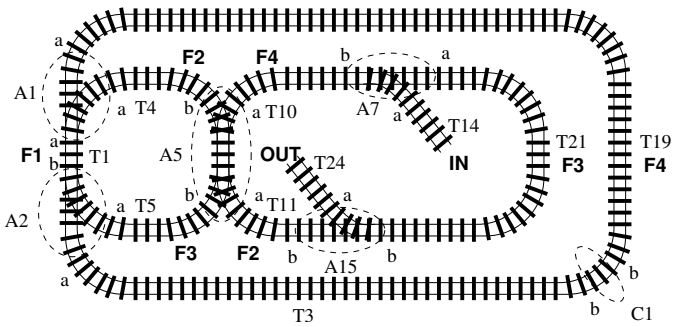

Fig. 1. Sample train circuit

connected, either through a CANOpen link or directly, to the turnouts as well as to photo-electric sensors placed at both ends of each section to detect when a train enters or leaves it. Orders to engines are sent to the Marklin Interface from a program on the PC via a RS232 link, this program acts as a bridge between the PLC and the engines and reads the speed and the direction of each engine in the memory of the PLC using the Modbus/TCP protocol. The PC also runs an application that performs reconfiguration of the circuit as well as starting the train using the ethernet link.

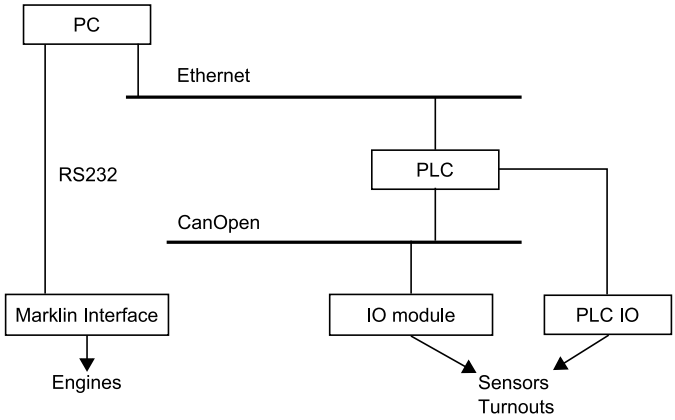

Fig. 2. Control/Command architecture

3. THE DESYRE LANGUAGE

DeSyRe [Lamotte et al., 2007] (french for Description of Reconfigurable Systems) is built upon model engineering concepts. This language is separated into two parts, according to figure 3. The architecture part is a model that holds the elements which constitute the system. Configuration models define how the architecture is used. A UML profile and a textual syntax have been defined to enable manipulation of DeSyRe models. Model transformations are used to go back and forth from the DeSyRe and the human readable form.

Architecture of the system is in turn divided into logical and physical parts. Logical part describes machining *functions* performed on the *products* and their association to form *function sequences* to obtain a finished product. Physical architecture describes the *resources* in the system and the links between them. Two kind of resources are considered: *stationary resources* and *transport resources*. *Connections* represent the potential transfer links between

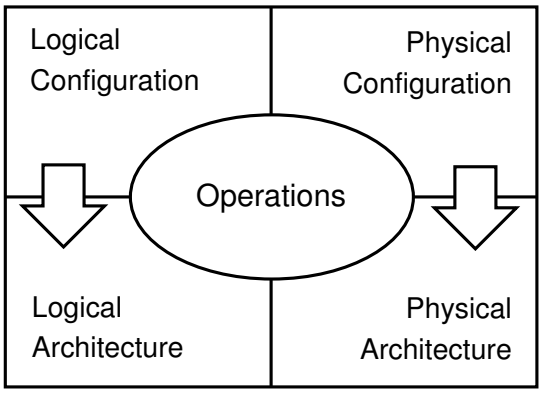

Fig. 3. Basic decomposition of the DeSyRe language

stationary resources, these connections are associated with the transport resources that can realize them. *Potential operations* complete the description of the architecture and links physical and logical parts by bridging functions and resources.

The configuration describes how the resources of the architecture are used to achieve a goal. Logical and physical aspect of the configuration can also be distinguished. The logical configuration is constituted of *function instances* and uses function sequences from the logical architecture. A function instance is the realization of a function on a product. It is performed on a resource through a *stationary operation*. Physical configuration is constituted of the resources taken from the logical architecture and *transfer sequences*. Transfer sequences are used to describe the transfer from one resource to another. They are composed of nodes which may either be *transporter nodes* or *controller nodes*. Transporter nodes are the realization of a connection using a given transport resource while controller nodes describe the routing from one port to another performed by a stationary resource. *Operations* defined in the configuration links both logical and physical configuration as well as architecture and configuration. There are two kinds of operations : *stationary operations* are instances of a potential operations, whereas *Transfer operations* link transfer sequences with a function instances. Operations are ordered into *operation sequences* that realize a function sequence from the architecture.

A simple architecture, in the manufacturing domain may be a two machines ($M1$, $M2$) system with a robot (R). This system performs, for instance, the $F1$ and $F2$ functions. The physical architecture for this system would consist in the two machines, the robot and the description of the transfer capabilities of the robot. The logical architecture would define the two functions and their organization into function sequences such as $S1 : F1, F2$. Some operations should also be defined for that system, if $M1$ is the support for $F1$ then an operation should be added to the architecture to link the two elements.

A configuration for this architecture would tell how its elements are used. In a configuration performing $S1$, the functions $F1$ and $F2$ should be instantiated. As for the physical configuration, a transfer sequence would be created to go from $M1$ to $M2$ using the robot. The logical and physical configurations are linked together by the

operation sequences ordering the operations that are used for the realization of $S1$.

4. MODELLING THE CIRCUIT WITH DESYRE

DeSyRe is a set of meta-models along with their representation used for the description of reconfigurable systems. Using this language, a train circuit can be described in many ways, depending mainly on the chosen granularity level. As the focus has been set on the machine and coordination levels of the system, it seemed judicious to give the priority on the cooperation between elements of the circuit.

A circuit is itself physically composed of sections and turnouts. To travel from a section to another, the train must go through turnouts and sections. The minimal transfer from a section to another uses one turnout. We decided to describe sections as stationary resources and turnout as transport resources that enable the transfer from a section to another, this transport resource is associated with one connection for each transfer it can perform between sections. This physical description of the system allows to describe all possible transfer between sections. Figure 4 is a representation of the studied train circuit, it features the description of section as stationary resources and turnouts as transport resources. One will note the special case for `C1`, which separates the `T19` and `T3` section, and which we decided to model in the same way as turnouts . Oddly, engines do not appear in the physical part of description. In fact we decided that physical description of the circuit would only represent its structure. Engines belong to the dynamic part of the system and are described in the logical architecture.

Logical part describes the functions and function sequences that can be carried on the physical part. These functions are performed on products that are mobile and can go from a stationary resource to another, which is what the engine does on sections as it travels from one to another. Once on a section, a real train undergoes changes, a passenger train at a station takes and leaves passengers for instance. So a train is modeled as a product that undergoes functions on the sections, the objective of the train is to complete function sequences by traveling through section on which described functions can be performed. Four functions will be used in the example, named `F1`, `F2`, `F3` and `F4`. These functions are used in two sequences : `S1 : F1, F2` and `S2 : F2, F3, F4`.

Potential operations have been designed to assign functions to the sections described in the physical architecture. Seven potential operations are described in the architecture, namely `F1T1`, `F2T4`, `F2T11`, `F3T5`, `F3T21`, `F4T10` and `F4T19`. The stationary ressource and the function implemented by the operation can easily be determined from their name, `F1T1` for instance is the implementation of `F1` on `T1`.

The architecture itself cannot tell how the function sequences are realized. This is the role of a configuration, based upon operation definition. 15 configurations have been written for the train circuit to date. In the context of this paper, we designed a simple configuration described on listing 1. This configuration realizes the `S2` sequence on the inner loop of the circuit, which is constituted by the `T10`, `T11` and `T21` sections. For this configuration, and to keep it simple, we consider that engine may either be on the `T10` or `T11` section at the begining. Function instances havec been provided for the needed functions. Three transfer sequences are defined to provide operations for moving the train between sections. The operation sequences defined in this configuration are quite simple. The first makes the train realize the `S2` function sequence. The second one is used to reset the train and prepare it for starting the sequence again (moving it from `T10` to `T11`.

Listing 1. A configuration for the train circuit

```
-- F2F3F4 on the inner loop
configuration l5c3 {
  logical configuration {
    instance IF2 of F2;
    instance IF3 of F3;
    instance IF4 of F4;
    instance IFT of FT;
  }

  physical configuration {
    tr seq TS_T11T21 {
      trans T11T21 using A15;
    }
    tr seq TS_T21T10 {
      trans T21T10 using A7;
    }
    tr seq TS_T10T11 {
      trans T10T11 using A5;
    }
  }

  tr op TO_T11T21 : IFT using TS_T11T21;
  tr op TO_T21T10 : IFT using TS_T21T10;
  tr op TO_T10T11 : IFT using TS_T10T11;

  st op SO_F4T10 : IF4 using F4T10;
  st op SO_F2T11 : IF2 using F2T11;
  st op SO_F3T21 : IF3 using F3T21;

  op seq OS_S2 realize S2 : SO_F2T11, TO_T11T21,
      SO_F3T21, TO_T21T10, SO_F4T10;
  op seq Rst : TO_T10T11;
}
```

5. CONTROL/COMMAND CODE GENERATION

This section defines basic control/command code generation principles. First the generation framework is introduced. The model engineering tools used to perform the transformation are then described. Once this is done, each step of the transformation is detailed as well as the reconfiguration of the system.

5.1 The code generation framework

Code generation is performed from the description model of the system. It follows the framework introduced on figure 5. Once the system has been modeled using DeSyRe and has been analyzed using the analysis framework presented in [Lamotte et al., 2007], the elements are grouped into components as described in section 5.3. In our example, the components corresponds to the elements of the circuit as described in section 4. A control command code written in IEC61131-3 [IEC, 2003] can be associated to each component depending on his type (section or turnout), communication between these codes is set-up according to the relation between the components. As in our case, we want to make the code work on our target PLC which is a Schneider TSX, the IEC61131-3 code needs translation to be understood by PL7-Pro, the

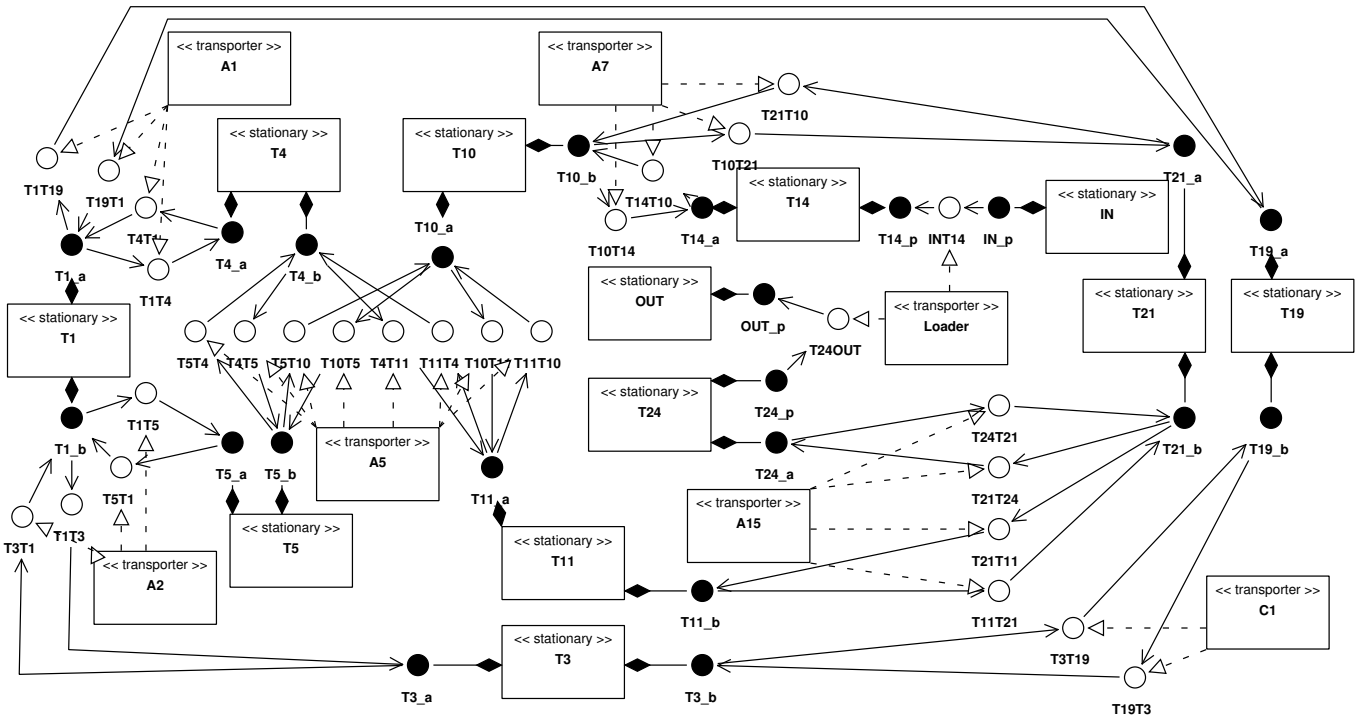

Fig. 4. Physical architecture for the train circuit described using the DeSyRe UML profile

programming tool for the TSX Premium family of PLCs, this translation consists in writing the code in the proper format and is described in section 5.5. It also consists in transforming the SFC code as defined into the IEC61131-3 into the equivalent grafcet code, used in PL7-Pro.

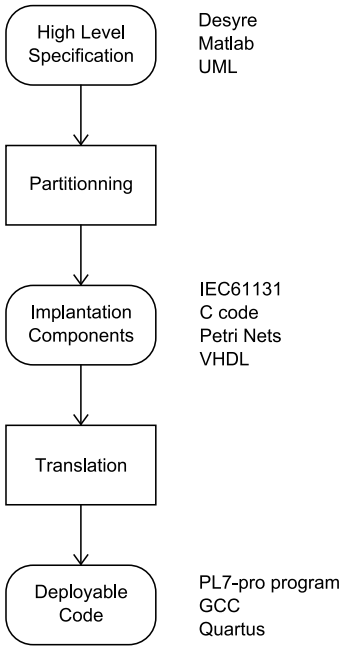

Fig. 5. Code generation framework

To achieve reconfigurability, it is necessary to provide reconfiguration mechanism. This can be done by applying the generation framework at runtime, to obtain a new code from a chosen configuration. A choice has to be made between generating the whole code off-line, preventing the system from being reconfigurated and generating it all on-line. In fact, it is possible, to generate only the code corresponding to the configuration. On the train circuit, one solution is to generate routing tables data from the configuration, which are uploaded to the PLC at runtime and used by the control/command code in the PLC to decide what to do. Both SFC code and routing tables are generated using the framework described on figure 5, but while the first is applied off-line, once and for all, the second is applied on-line.

5.2 Introduction to model engineering

Model engineering is used to implement the generation framework. It acts as a glue between the models. Models are defined by a meta-model and mapping between them are expressed into a model transformation language. Injectors and extractors are also used to import and export models to technical space (Petri nets, text files or XML for example), tools relevant to these technical spaces may then be used to get results from the models.

The model transformation tool used for these experiments is the atlas transformation language (ATL) [Bézivin et al., 2003], which is part of the AMMA platform and the Eclipse M2M project. ATL is a prototype developed at the University of Nantes by the ATLAS Team. AMMA provides ways to seize meta-models, to express mappings between them, to perform the transformations and use injectors and extractors.

An ATL transformation is composed of rules that define how source models elements are matched and navigated to create and initialize the elements of the target models. In

ATL, the navigation rules syntax is based on the object constraints language (OCL) [OMG, 2003].

Model engineering approaches enforces consistency between the different models used in the design and in the implementation of the system. Model manipulation is eased since model management is done by ATL. Another advantage of model engineering is that once a model has been defined by its meta-model, it can be used in other contexts.

The next subsections describe how model engineering is used for implementing the proposed framework.

5.3 Partitioning the model into components

The first step in the framework consists in slicing the model into components. This partitioning may be done according to several rules. A logical partitioning would lead into components to drive the realization of a given product. This step is performed using a model transformation.

For the train circuit example, we have applied a physical partitioning. This partitioning leads to the creation of one component for each physical element in the system (turnout or section). These elements are easily matched in the physical architecture model represented on figure 4 as each section is represented by a `Stationary Resource` and each turnout by a `Transport Resource`. For the code generation, the `C1` resource is considered as a turnout which only has two entries.

5.4 Taking I/O into account in code generation

In order to properly perform the transformation to PLC program, the link between elements defined in DeSyRe and the actuators and sensors must be made. These links are defined through another model called IOA (Input Output Association), which is mapped onto the DeSyRe model using the names of its elements. In our application, target for code generation is well defined and is a PLC. An IOA model is constituted of several `IO`, which associates one or several element to a IEC61131 variable. To enable more complex constructs, one can associate a PRL action written in the ST language, this action will be added to the PRL section of the SFC (which is executed just before the SFC in the PLC cycle), it can be used to call a function block for instance.

Listing 2 is an excerpt of the IOA definition used for the train circuit featuring the description of the link between a `Port` and a sensor, which can be accomplished either by directly using the IO name (the case for T1_a) or by using an action (for T19_b). In the latter, we had to access one bit of a word, it can also be noted that `T19_b` and `T3_b` share the same sensor. As for turnout command, an output from the PLC is associated to the connection it enables. Here, using `%Q3.1` the `A2` turnout will be set so that a train on the `T1` section will travel to section `T3`. The actual IOA definition for the circuits defines the 27 inputs or outputs controlled by the PLC.

Listing 2. Example IOA model description

```
io '%I2.2' { Port 'T1.T1_a' ; }

io '%M3' ['%M3 :=  %MW20:X3;'] {
  Port 'T19.T19_b';
  Port 'T3.T3_b';
}

io '%Q3.1' {
  Connection 'T1T4';
  Connection 'T4T1';
}
```

5.5 Generating the code for each component

Partitioning rules have been devised and that elements in the model are associated with their corresponding physical elements. Now a model transformation has to be written to generate the code for each component. Communication primitives enabling communication between these components must also be generated. The IOA model is used during the code generation to write the ST phrases that reads inputs and write outputs.

For the train circuit, the control/command code of the elements is described into the SFC language. Communication between the SFCs is the basis of the reservation mechanism that permits the train moves along the circuit. These moves are encoded into routing tables, which are generated from the configuration and are uploaded into the PLC on each reconfiguration of the circuit. To read the routing tables, components need to know which operation sequence an engine is currently performing, this information is stored in the memory of the PLC and is set by a program on PC.

Figure 6 describes the cooperation mechanism used for moving an engine from a section (`s1`) to another section (`s2`) through a turnout (`t`). First, `s1` looks up its routing table and reads that the train should go through the `t` turnout. It asks `t` for a transfer. `t`, in turn looks up its table and reads that the train should move to `s2`. It sets itself up and waits for `s2` to be ready. When both sections are ready, `a` informs `s1` that the transfer can begin. The `s1` component starts the engine. As soon as it leaves `s1` is ready to accept a new engine. Once the train arrives on `s2`, as the transfer is finished, `t` is ready to perform new transfers. `s2` stops the engine a consults the routing table to decide what to do.

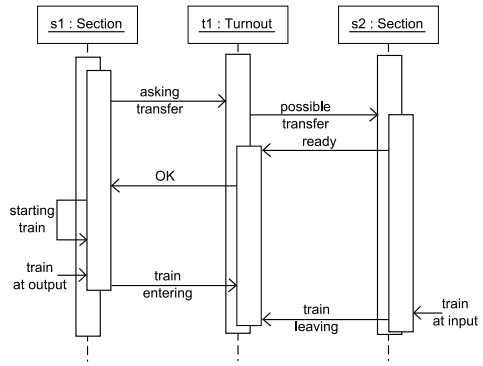

Fig. 6. Cooperation between elements in the circuit to drive the engines

5.6 Configuration of the circuit

IEC61131 code generated for the train circuit needs routing tables in order to work. These routing tables are obtained from elements in the configuration model. Routing

tables generation follows the framework defined in this section, they are also obtained from a physical decomposition of the system, as one table is generated out of each resource.

A routing table is composed of several line. Each line associates three element : the transfer sequence, the input port and the output port. When a train is on a resource (section or turnout), this resource can obtain the port by which the train should leave from the transfer sequence it is currently performing and the port from which it entered.

Table 1 gives a sample routing table, generated out of the l5c3 configuration described on listing 1. This table only consists in one line, as A5 is only used in on transfer sequence : TS_T10T11, input and output port are obtained from the T10T11 connexion, defined in the physical architecture represented on figure 4.

Transfer sequence	Input port	Output port
TS_T11T10	T10.T10_a	T11.T11_a

Table 1. Routing table for the A5 section in the considered configuration

Routing tables are sent to the PLC memory from the PC by directly writing data into the PLC memory.

Configuration not only defines transfer sequences that permits the generation of the routing tables, but also operation sequences. These operation sequences are lists of operations which can either be a transfer operations (associated with a transfer sequence) or a stationary operation. The operation sequences are also transfered to the PLC memory and are used by the SFC associated to turnouts to advance in operation sequences.

6. RESULTS

The PL7-pro program generated for the train circuits is 4298 lines long. It consists in 317 steps and 362 transitions.

This code successfully drives engines on the circuit. We have been able to set-up reconfiguration scenarios. One of these scenarios starts with a configuration performing the S2 function sequence (F2, F3, F4) on the inner loop and the outer loop without changing direction. After a failure on the T21 section, a new configuration is chosen and sent, which uses the small and outer loops. When another failure occurs on the T3 section, the same loops are used but the engine must changing direction on the T19 and T1 sections to avoid T3.

7. CONCLUSION

This article has shown how model engineering techniques could be used to generate control/command code from a higher-level description of the system. This generation provides non negligible benefits.

Design is performed on a high-level model of the system. Using this model ensures the consistency between analysis performed on the model and its implementation, since the same input is provided to both activities. Then, it enables to easily apply modifications either on the physical system, or on the functional principle of the components.

Future work concerning code generation will focus on defining a component model. Modeling the control/command architecture is also necessary to be able to deploy the components on it. This description can be done using the DeSyRe language. When a model of the control/command architecture exists, it should be possible to expand the IOA definition so that it is less linked to a technology.

As for reconfiguration, for now the configuration is chosen from a predefined pool. Future work will focus on the automatic generation of the configuration.

REFERENCES

Jean Bézivin. On the Unification Power of Models. *Software and System Modeling (SoSym)*, 4(2):171–188, 2005.

Jean Bézivin, Grégoire Dupé, Frédéric Jouault, Gilles Pitette, and Jamal Eddine Rougui. First experiments with the ATL model transformation language: Transforming XSLT into XQuery. In *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, 2003.

IEC. *Programmable Controllers. Part 3 : Programming Languages*, 2003. IEC 61131-3.

F. Lamotte, P. Berruet, and J.-L. Philippe. Evaluation of Reconfigurable Manufacturing Systems configurations using tolerance criteria. In *Proceedings of the IEEE IECON Conference*, Paris, 2006.

F. Lamotte, P. Berruet, and J.-L. Philippe. Using model engineering for the criticality analysis of reconfigurable manufacturing systems architectures. *International Journal on Manufacturing Technology and Management, Special Issue on Manufacturing Under Changing Environment*, 11(3/4), 2007.

M.G. Merhabi, A.G. Ulsoy, and Y. Koren. Reconfigurable Manufacturing Systems: Key to Future Manufacturing. *Int. Journal on Intelligent Manufacturing*, 11:403–419, 2000.

OMG. Response to the UML 2.0 OCL RfP., 2003.

K. Thramboulidis, G. Doukas, and A. Frantzis. Towards an implementation model for fb-based reconfigurable distributed control applications. In *Proceedings of the seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'04)*, 2004.

Brigit Vogel-Heuser, Daniel Witsch, and Uwe Katzke. Automatic code generation from a uml model to iec 61131-3 and system configuration tools. In *2005 International Conference on Control and Automation (ICCA2005)*, Budapest, Hungary, June 2005.