Platform-Independent Specification of Model Transformations @ Runtime Using Higher-Order Transformations

Michael Schlereth[1,2], Tina Krausser[3]

[1]Siemens AG, Industry Sector, I DT MC R&D 7 3
Frauenauracher Str. 80, D-91056 Erlangen
[2]Technische Universität Darmstadt, Institut für Datentechnik,
Fachgebiet Echtzeitsysteme, Merckstr. 25, D-64283 Darmstadt
michael.schlereth@siemens.com

[3]Lehrstuhl für Prozessleittechnik der RWTH Aachen,
Turmstraße 46, D-52064 Aachen
tina.krausser@plt.rwth-aachen.de

Abstract: Model transformation specifications are currently bound to a specific transformation execution engine, typically executed on desktop systems. This paper presents a novel approach to transform platform-independent model transformation specifications into platform-specific model transformation specifications, which can be executed for example by the runtime systems of process control systems employed in plant automation. For that purpose, the concept of higher-order transformations was adapted to the transformation between platform-independent and platform-specific models of model transformations.

1 Introduction

Process control systems controlling plants e.g. for chemical processes include the runtime of models from different engineering disciplines (see Figure 1). The plant engineering model, based on process and instrumentation diagrams (P&ID), is used for example by the batch planning and process diagnosis runtime system. The HMI runtime controls the operator panels used to operate the factory. The runtime of the automation devices controls the sensors and actors of the plant equipment, such as vessels, pumps and valves.

Process plants are operational for many decades executing continuous manufacturing processes, which usually can't be interrupted. Therefore, process control systems are typically updated and reconfigured at runtime. This means that the plant continues operation while the runtime system is modified. Reasons for such a modification might be an extension of the plant, a process redesign or the replacement of faulty devices. To simplify these modifications of the running plant, process control systems such as ACPLT [Le11], which is used as an example in this paper, abandon the separation of engineering and runtime by keeping all engineering models in a common executable runtime database as shown in Figure 1. Since the models are modified independently by

engineers working in different disciplines, a rule-based system, called ACPLT/RE, runs as part of the runtime and aligns the other models in real time if one of the engineering models is changed. To be executable in the process control runtime system, all engineering models together with the model of the reconciliation rules are specified by the automation controller programming languages defined by IEC 61131-3 [In03].

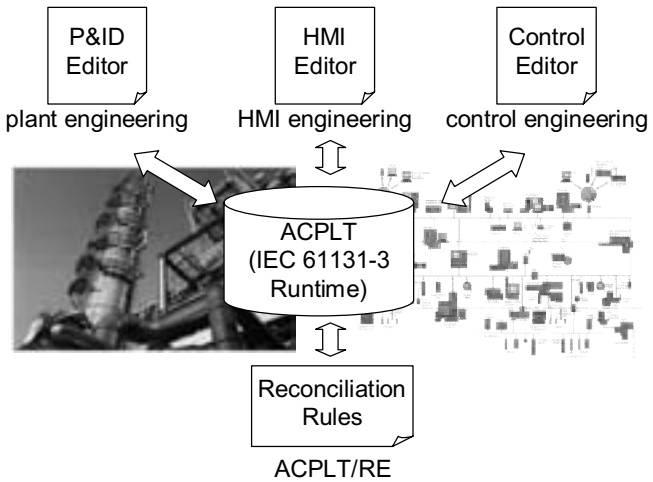

Figure 1: Runtime environment of a process control system

In this paper, a novel approach is presented for the specification of these model reconciliation rules executed at runtime. The concept of the PIM/PSM approach of the model driven architecture [MM03] is extended and transferred to the platform-independent specification and platform-specific execution of model transformations. This new concept is used to execute model transformation languages, which are currently only available for desktop applications as model transformations @ runtime for process control systems.

2 Application Scenario

This section is split in two parts. The first part describes the engineering process of the application scenario for model transformations @ runtime from a platform-independent view, while the second part introduces the process control system ACPLT used as the specific implementation platform [Le11].

2.1 Engineering of Process Control Systems

The application example used in this paper demonstrates the engineering process of a process control system (PCS). As an example for this engineering process, a pumping

plant designed for teaching purposes is used in this paper. As part of this engineering process, a plant engineer creates a model of the pumping process, visualized as a P&ID (piping and instrumentation diagram) shown to the left in Figure 2. Plant assets used by the plant engineer for the pumping process includes vessels, pumps, and valves, for example. The task of the control engineer is to create HMI (human machine interface) screens for the plant operation based on the assets defined by the plant engineer (shown to the right in Figure 2). Both engineering models - the plant engineering model and the HMI engineering model - are stored together with other engineering models of the pumping plant (e.g. the engineering model of the control functions) in the runtime environment of the process control system (PCS) at the pumping plant site.

For a lot of the assets in the plant model, a HMI symbol must be created on the HMI screen to operate this asset: a vessel HMI symbol for the vessel plant asset, a pump HMI symbol for the pump plant asset, and a valve HMI symbol for the valve plant asset. It is the task of the plant engineer and of the control engineer to keep both models consistent (indicated by the arrows in Figure 2): If a plant asset is created in the process model, a HMI symbol must be created in the HMI model. If a plant asset is deleted in the process model, the corresponding HMI symbol must also be deleted. This reconciliation is a tedious manual task, which is especially error prone in a parallel engineering process, where the plant model and the HMI model are engineered simultaneously. Therefore, a rule-based system called ACPLT/RE [KQE11] was developed, which automates the reconciliation between plant engineering models. ACPLT/RE is executed as part of the runtime of the process control system (PCS), because plants in the process industry, which have been operating for many decades, are typically updated on site while the plant continues to operate. Rule-based engineering for automation systems is used to automate engineering processes within process control engineering (PCE) [SE06].

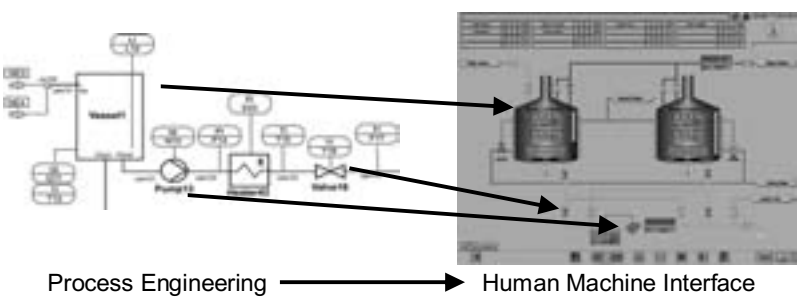

Figure 2: Application example: engineering a pumping plant

2.2 The ACPLT Process Control System

The rule-based engineering system ACPLT/RE together with the plant engineering model RIVA and the human machine interface model ACPLT/HMI is stored and executed by the ACPLT process control system. All three models are based on the ACPLT/FB function block library, which implements an IEC 61131-3 [In03] compliant

process automation system. A commercial implementation named iFBSpro is provided by [LT04].

The ACPLT/FB object model consists of two main types: function block instances (FB_Instance) and function block links (FB_Link) as shown in Figure 3. A function block is a programmable controller programming language element consisting of a data structure with input, output, and internal variables [In03]. Instance values of the function block data structure are stored in ACPLT/FB by the FB_VariableValue class. The three engineering models of the application example are based on the OV_Library called gssffa for plant engineering, hmiRIVA for HMI engineering, and reSFCbased for rule-based engineering.

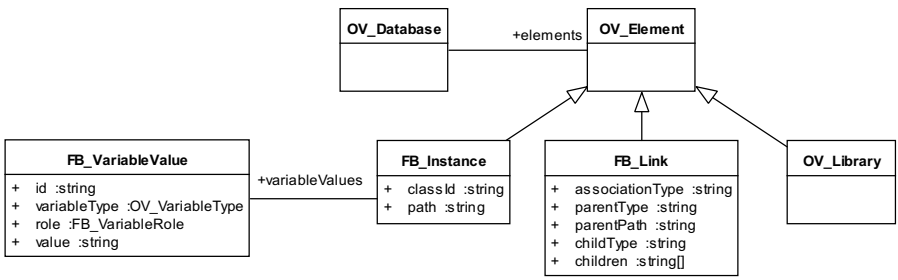

Figure 3: The ACPLT/FB object model [LT04]

3 Specification of Model Transformations @ Runtime

This section presents the novel concept of platform-independent model transformation specifications (PIM-MT) and platform-specific model transformation specifications (PSM-MT), which can be applied during the design phase and @ runtime. The current usage of model transformation technology focuses on the platform-independent model and the platform-specific model for a system implementation [MM03] but not on the platform of model transformation execution systems itself. In the process plant application scenario it is advantageous to use a platform-independent model transformation model for two reasons:

- A platform-independent model transformation language allows for a more compact and formally verified specification than the model transformation specification @ runtime by ACPLT/RE in IEC 61131-3. The same platform-independent specification can be used during the design phase and at runtime using different transformation engines.
- Over the lifetime of a plant, the model transformation system @ runtime might change due to the fact that a new process control system has been installed. A platform-independent model transformation language allows for the migration of the model transformations to a new runtime system.

Therefore, the PIM/PSM approach is extended to include the transformation of model transformations as shown in Figure 4, which shows PIM and PSM levels (but not M2

and M1 levels). This transformation is called a higher-order transformation (HOT), since it is a transformation of transformations [Ti09]. The HOT transformation specification describes the transformation from an ATL transformation specification metamodel (M2) to an ACPLT/RE rules specification metamodel (M2). The execution of this HOT transformation specification transforms a specific ATL transformation specification (M1, e.g. the RIVA to HMI transformation) to a specific ACPLT/RE rule execution specification (M1). The reverse model transformation specification describes the transformation from an ACPLT library (M3) to the ECORE metamodel (M3). The execution of this transformation specification transforms a specific ACPLT library (M2, e.g. the HMI library) to a specific ECORE metamodel (M2, e.g. the HMI metamodel). In our new approach, the concept of HOT has been extended so that it can be used for plain model transformations as well as for model transformations of model transformations. A special characteristic of this HOT is that it is not only directed from the PIM to the PSM but that it also includes a transformation in the reverse direction from the PSM to the PIM with respect to the referenced metamodels and libraries. This aspect will be explained in detail in Section 4.

The next two subsections describe in detail the platform-specific model transformation @ runtime and the related platform-independent transformation @ desktop, running outside the process control system.

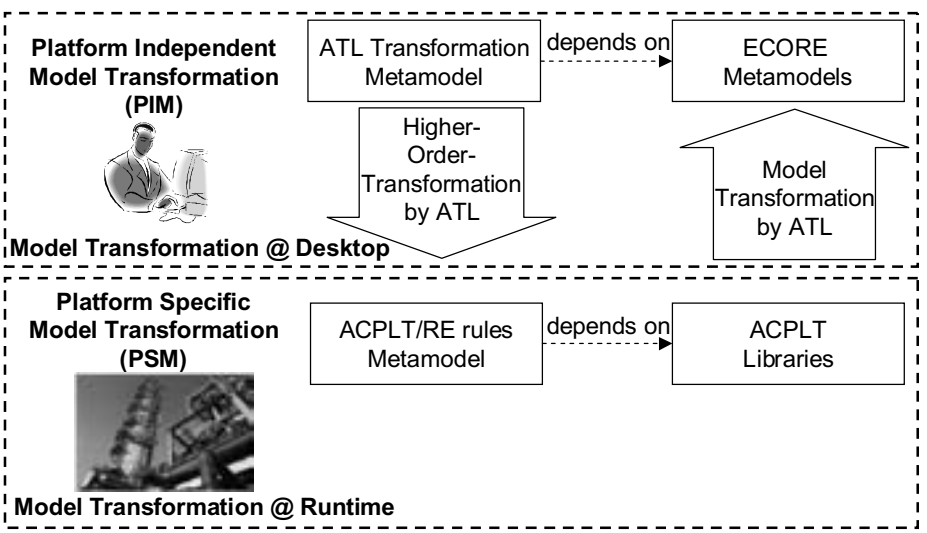

Figure 4: PIM to PSM higher-order transformation

3.1 Platform-Specific Specification of Model Transformations @ runtime

The engineering rule for the ACPLT engineering elements introduced in Section 2.2 is as follows: For each gssffa/Unit create a hmiRIVA/Unit and set the UnitType variable of hmiRIVA/Unit to the same value as the UnitType variable of gssffa/Unit. This kind of

rule can be executed by the rule-based engineering system ACPLT/RE as part of the plant runtime system. ACPLT/RE is implemented as an OV_Library in the same way as the plant engineering and the HMI engineering model. The advantage of this common implementation of all engineering models in a process control system is that an automation engineer, who is familiar with IEC 61131-3 programming languages, can work with all of these models for plant engineering, operation, and maintenance.

To specify the rule above, ACPLT/RE provides the function blocks reSFCbased/RuleExecControl for the rule definition, reSFCbased/RuleChain for the condition and the conclusion of the rule and reSFCbased/ShadowFB for the model pattern used by the condition and the conclusion of the rule. The use of the ACPLT/RE functions to specify the rule above is shown in Figure 5.

The rule is defined by the function block instance rivaUnit2HmiUnit (shown at the top of Figure 5). This function block instance aggregates the condition and conclusion function block instances (shown with their children as two blocks below the rule in Figure 5).

As mentioned in the description of the ACPLT/FB object model, this aggregation is not visible in the typed object model but in the string values of the path attribute of the function block instances (e.g. /TechUnits/Mike/Rules/rivaUnit2HmiUnit/Condition). The condition defines a pattern, which must be found in the engineering model to execute the rule rivaUnit2HmiUnit. In ACPLT/RE, patterns of function blocks are defined by so called shadow function blocks (classId reSFCbased/ShadowFB). A shadow function block defines the library and the class of a function block instance, which should be matched. In Figure 5, the shadow function block instance named rivaUnit matches function blocks from the library gssffa with the classId Unit. The shadow function block can define additional constraints such as variable values in addition to the classId. For simplicity, these constraints are omitted in our example.

For each match of the condition, the conclusion of the ACPLT/RE rule is applied. The conclusion also uses shadow function block instances to define a pattern, which is generated in the engineering model. In the example, the conclusion hmiUnit generates a function block instance from the library hmiRiva with the classId Unit. Also the conclusion can set additional variable values, which is also omitted for simplicity in Figure 5.

In ACPLT/RE, the simple example for an engineering rule formulated at the beginning of this section, which maps plant engineering units to HMI engineering units, totally consists of 41 function block programming elements: 16 function block instances and 25 function block link elements (the instance diagram of function block link elements for the ACPLT/RE rule has been omitted in this paper for to keep it short).

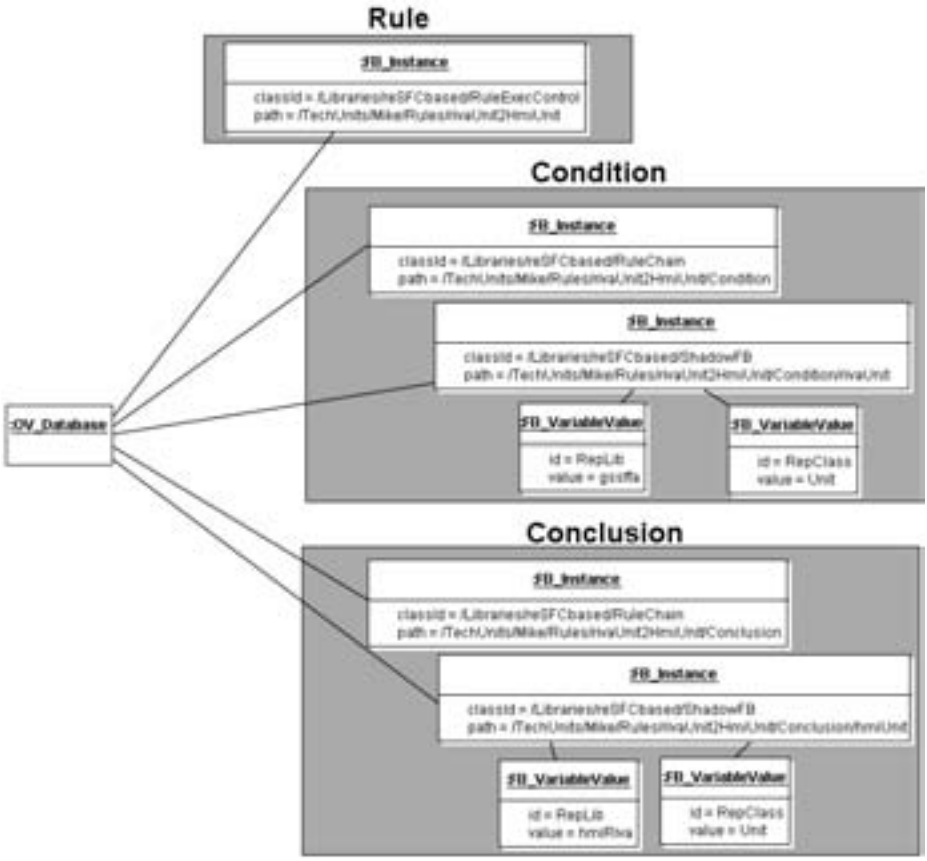

Figure 5: Simple ACPLT/RE rule example (function block links omitted)

The use of function block programming according to IEC 61131-3 for engineering definitions by ACPLT/RE has advantages as well as disadvantages. The advantage of function block programming is the compatibility to the runtime system of a process control system in a plant: the engineering rules can be executed online in the operational plant automation system without the need to shut down the plant for extensive engineering data transfer. The typical commissioning use case for automation systems, where automation engineers with PLC programming background must get the automation system running, is also well addressed using the commonly known IEC 61131-3 programming paradigm. The declarative approach of the ACPLT/RE rule definitions is easy to understand.

A disadvantage of ACPLT/RE is the high amount of function block programming elements, which are required to define even simple rules. A more abstract syntax than IEC 61131-3, which is transformed to ACPLT/RE would ease the definition of engineering rules. Another problem is the dependency on the target system: Plant

engineers typically must support different automation system providers according to customer requirements. If a plant operator does not use ACPLT technology but for example the SIEMENS PCS7 process control system, the rules should also be able to be executed on this process control system. Therefore, a more abstract and compact rule definition language, which can be transformed to ACPLT/RE, would ease the application of rule-based engineering.

3.2 Platform-Independent Specification of Model Transformations

While ACPLT/RE considers the rules required to automate our engineering scenario of Section 2.1 as the primary artifacts, the model transformation community would consider the engineering models for plant control engineering and HMI engineering as the primary artifacts and would talk of a model transformation from the plant engineering model to the HMI engineering model to automate our engineering scenario.

For industrial use, a mature and proven model transformation language is required. Therefore, the ATL transformation language, introduced by [Jo06] and implemented on the Eclipse platform was selected. ATL transformations are compiled and executed by an ATL-specific virtual machine, which transforms a source Ecore model to a target Ecore model (Ecore is a model definition language introduced by the Eclipse modeling platform).

An ATL model transformation is specified by a set of rules, which specify the mapping of source elements to target elements. The ATL rules are aggregated as elements of type ModuleElement in an ATL module, which is the container of all ATL rules. The declarative ATL rules considered in this paper are a special ModuleElement called MatchedRule. Each MatchedRule consists of an InPattern and an OutPattern. The InPattern is specified by an OclExpression. The object constraint language (OCL) was originally designed to describe expressions on UML models and was reused by ATL for data types and declarative expressions. The OutPattern is specified by a set of bindings, which also use OCL to assign values to attributes of the created target model elements. The detailed ATL model referred to is described by [Ti09]

The ATL transformation language rule definitions are based on a textual syntax, which allows for more abstract and compact rule definitions than ACPLT. The development of ATL rules is well supported on the integrated development environment Eclipse, e.g. with context sensitive editors and team support. In addition to the textual syntax, ATL provides an Ecore model for the rule definitions, which allows the processing of ATL rule definitions by higher-order transformations [Ti09]. In contrast to the weak type definitions of ACPLT/RE function blocks based on string variables (cf. RepLib and RepClass variables in Figure 5) ATL rules support strong typed pattern elements. This strong type supports avoids rule errors due to invalid types at compile time of the ATL rules and not at run time as with ACPLT/RE.

The disadvantage of ATL is that the transformations of models at runtime is not supported: a typical ATL transformation setup assumes that the source and target models are exported from the runtime environment of a process control system to the Ecore

model format and after transformation are re-imported to the runtime system instead of supporting intertwined execution of the plant control software and its transformations. Furthermore, the ATL transformation engine is available only on the Java platform, which would require an additional Java-compatible industrial PC device in typical plant automation system setups.

4 Higher-Order Transformation from ATL model transformations to ACPLT/RE rules

This section presents in detail the higher-order transformation between ATL model transformation specifications and ACPLT/RE model transformations @ runtime as introduced in Figure 4. For the process automation application scenarios the combination of the advantages of ACPLT/RE and ATL would ease the definition and execution of engineering rules in automation systems, e.g. the reconciliation between plant engineering and HMI engineering models as described in Section 2.1. ACPLT/RE provides the runtime support for rule execution on the process control system (PCS) and is easy to understand for automation engineers e.g. when commissioning plant extensions. ATL provides type-safe and compact rule definitions, which are independent of the execution platform.

Therefore, a transformation from ATL rule definitions, engineered in the Eclipse platform, to ACPLT/RE rule definitions, executed by the ACPLT plant control system, was developed. With that transformation, it is possible to define the plant engineering to HMI engineering transformation by ATL, but execute the transformation by ACPLT/RE.

In ATL, the engineering rule example shown in Figure 5 is expressed as shown in Figure 6. Just as the ACPLT/RE rule, for every plant asset of type `"gssffa::Unit"` (which could be a vessel, a pump or a valve according to the UnitType attribute), the ATL rule creates an HMI engineering element of type `"hmiRIVA::Unit"`. The OCL expression `"rivaUnit.path.startsWith"` in the „from“ clause of the rule selects the context of the plant asset within the plant structure. Within the “to” clause of the rule, the UnitType value of the newly created HMI engineering element is set to the same value as the UnitType value of the corresponding plant asset.

The approach presented by this paper transforms ATL rules, such as the rule presented in Figure 6, to ACPLT/RE rules, such as the rule presented in Figure 5. This transformation was implemented by ATL higher-order transformations between the ATL and ACPLT/RE languages. ATL was chosen for the higher-order transformation because both ATL and ACPLT/RE languages can be specified by Ecore models and because every additional language would increase the complexity of the engineering system. The Ecore model of ATL rules is provided by the ATL implementation itself. The Ecore model of ACPLT/RE was developed as part of the work presented by this paper with the help of a XTEXT [Th11] grammar for the textual export format FBD of ACPLT.

```
-- @path MMRIVA=/atl2fbd/model_lib/rivaBalance.ecore
-- @path MMHMI=/atl2fbd/model_lib/hmi.ecore

module riva2hmiTyped;
create OUTHMI : MMHMI from INRIVA : MMRIVA;

rule rivaUnit2hmiUnit
{
       from
              rivaUnit : MMRIVA!"gssffa::Unit"
              ( rivaUnit.path.startsWith(
                     '/TechUnits/RIVA/IC001/IC001/IC001/TU10'))
       to
              hmiUnit : MMHMI!"hmiRIVA::Unit"
              ( path<-'/TechUnits/Mike/Simotion'+rivaUnit.name
              , UnitType <- rivaUnit.UnitType )
}
```

Figure 6: Plant asset to HMI element engineering rule

According to the analysis of the authors, ATL and ACLT/RE are both structured in four areas as shown in Figure 7. The rule language defines the relationship between engineering models. The engineering models are defined by rule language-specific system models. The pattern language is used by the rule language to define the model elements selected or constructed by the rule language. Finally, the inter-rule execution control determines the execution order of multiple rules and resolves model element dependencies between rules. The implementation presented here covers the transformation of the rule language, the system model, and a small part of the pattern language as required by the process control engineering scenarios presented in Section 2.1. More complex pattern language constructs and inter-rule execution control are considered as future extensions of the work presented here.

	ATL	**ACPLT/RE**
Rule Language	ATL matched rules	ACPLT/RE rules
System Model	Ecore metamodels	ACPLT libraries
Pattern Language	OCL	IEC 61131-3 function blocks (FB)
Inter-Rule execution control	ATL traceability	IEC 61131-3 sequential function charts (SFC)

Figure 7: Comparison of the structure of the ATL transformation language and the ACPLT/RE rule language

4.1 Transformation of the System Model and the Rule Language

The ATL rule language consists of two elements: ATL matched rules and Ecore metamodels referenced by these rules (see Figure 7). Therefore, the transformation of the ATL rule language into ACPLT/RE consists of two ATL transformations in different directions (see Figure 4):

- An ATL transformation from ATL to ACPLT/RE to transform the ATL matched rules to ACPLT/RE rules. (1:n transformation from platform-independent representation to different specific platforms).
- An ATL transformation from ACPLT/RE to ATL to transform the ACPLT/RE libraries of the engineering models to Ecore metamodels, which can be used by ATL matched rules. (n:1 transformation from different specific platforms to platform-independent representation).

The ATL transformation, which is required to transform the ATL rule language into ACPLT/RE, is a higher-order transformation (HOT) and is implemented as the ATL module `atl2fbd.atl`. A higher-order transformation uses a transformation language to transform the transformation language itself: in our case an ATL transformation transforms the ATL transformation language into ACPLT/RE. Therefore, it is a transformation from the ATL metamodel `MMATL` to the ACPLT/RE metamodel `MMFBD`, which is represented as an XTEXT grammar for FBD files, the data exchange format of ACPLT/FB. Figure 8 shows an exemplary part of the `atl2fbd.atl` higher-order transformation: the transformation from an ATL matched rule `MMATL!Rule` (the "from" clause in Figure 8) to an ACPLT/RE `RuleExecControl` function block, which is specified as an `MMFBD!FB_Instance` with a classId `/Libraries/reSFCbased/ RuleExecControl` (the "to" clause in Figure 8). An ACPLT/RE rule also consists of an instance of the condition and conclusion function blocks, which are created along with the `RuleExecControl` function block instance in the "to" clause of the ATL higher-order transformation. In addition to these function block instances, ACPLT/RE also requires the creation of function block connections and links to the task system of the process control system for its elements. In the FBD metamodel, these elements have a more complex creation pattern, which can be reused in different rules of the higher-order transformation. Therefore, this creation pattern is implemented for reuse by the ATL called rule, which is called `createTaskLink`.

The second transformation, the transformation of the plant engineering metamodel (ACPLT library gssffa) to the HMI engineering metamodel (ACPLT library hmiRiva) in the reverse direction to the higher-order transformation, is implemented by the ATL transformation `ovmString2ecore.atl`, which generates the Ecore metamodels `rivaBalance.ecore` and `hmi.ecore`.

The ACPLT libraries are defined by description files in a textual format called OVM, with a similar syntax as used by the FBD format defined by ACPLT/FB (see Section 2.2). Therefore, the same XTEXT parser could be used for FBD and OVM files. In contrast to FBD, OVM doesn't include instance information for objects, but instead type information. The `ovmString2ecore.atl` transformation is generic and

transforms every OVM file into an Ecore model depending on its run configuration. Currently, the only limitation of the `ovmString2ecore.atl` transformation is the fact that the OVM `#include` statement is not supported in the XTEXT grammar. Therefore, library definitions that are included must be copied into a combined file before running the `ovmString2ecore.atl` transformation.

With the help of the two generated metamodels `rivaBalance.ecore` and `hmi.ecore`, it is possible to specify the transformation from the plant engineering model to the HMI engineering model as an ATL matched rule, as shown at the left side of Figure 4 (a rule example was presented in Figure 6).

```
rule rule2ruleexec
{
    from
      s: MMATL!Rule
    using
    {
       ruleFbName : String =  thisModule.ruleDomain+s.name;
    }
    to
      t: MMFBD!FB_Instance
      (
       path <- ruleFbName
       , classId <- '/Libraries/reSFCbased/RuleExecControl'
      )
      ,condition: MMFBD!FB_Instance
      (
       path <- ruleFbName+'/Condition'
       , classId <- '/Libraries/reSFCbased/RuleChain'
      )
      ,conclusion: MMFBD!FB_Instance
      (
       path <- ruleFbName+'/Conclusion'
       , classId <- '/Libraries/reSFCbased/RuleChain'
      )
    do
    {
      thisModule.createTaskLink('RuleChain', ruleFbName);
-- additional createTaskLink ommitted
    thisModule.createFbConnection(ruleFbName,'Start1',
ruleFbName+'/Condition', 'Command');
-- additional createTaskLink ommitted
    }
}
```

Figure 8: ATL rule, which transforms an ATL rule into an ACPLT/RE rule

4.2 Transformation of the Pattern Language

The second part of the ATL transformation language, the pattern language (see Figure 7), is based on the OCL language and is required to compute attribute values either for pattern matching or for pattern creation. The computation of attributes is called "binding" in ATL. For the rule-based engineering application in process control engineering, the focus is on the application of rules with simple bindings on a huge number of objects. This is done to avoid faults due to tedious manual engineering. Therefore, the OCL string comparison expression "startsWith" and value assignments already cover a notable application scope (see Figure 6). These two OCL expressions are currently implemented as part of the ATL higher-order transformation. Figure 9 shows an excerpt of the HOT rule, which transforms the variable binding of the "to" clause of the platform-independent ATL rule shown in Figure 6 to ACPLT/FB function block elements. The rule is executed for each `MMATL!NavigationOrAttributeCallExp` and creates a function block instance of type `/Libraries/tmtlib/getVariable`. According to the ACPLT/FB metamodel (see Figure 3), the variables of this function block instance must be initialized by the creation of six `MMFBD!FB_VariableValue` elements.

```
rule BindingNavigationOrAttributeCallExp
{
       from
              s: MMATL!Binding
       (s.value.oclIsTypeOf(MMATL!NavigationOrAttributeCallExp))
       using  {  } -- local variables
       to
              -- create getVariable function block
              t: MMFBD!FB_Instance
              (      path <- getVariableFbName
                     ,  classId <-
                            '/Libraries/tmtlib/getVariable'
                     , variableValues <-
                     OrderedSet{t_varname,t_iexreq,t_actimode} )
              , t_varname: MMFBD!FB_VariableValue
              (      id <- 'varname'
                     ,role <- 'INPUT'
                     ,variableType <- 'STRING'
                     ,value <- '"'+sourcePropertyName+'"' )
    -- continued with 5 more MMFBD!FB_VariableValue elements
    -- and ATL do block
}
```

Figure 9: ATL rule, which transforms an OCL binding expression into ACPLT/FB blocks

The experience gained from implementing of the transformation from the OCL language to IEC 61131-3 function blocks by an ATL higher-order transformation indicated that it will be difficult to specify the higher-order transformation for more complex OCL statements because the OCL language and the standard function blocks of IEC 61131-3 language are very different in their language structure. This problem can be resolved, if

ACPLT/RE would be extended by a function block helper library, which better matches the OCL expression syntax than the currently available IEC 61131-3 function blocks.

5 Related Work

The integration of engineering models of process control systems was investigated by [BHW05]. The focus of this work was on the offline integration of desktop application and did not address models @ runtime nor the PIM/PSM approach with higher-order transformations presented here. The classification of transformations by [MV05] considers the PIM to PSM transformation as a vertical transformation. The term "higher-order transformation" is used for transformations of model transformations on the same platform but not in relationship to a vertical transformation as presented in this paper. The higher-order transformation support of ATL was introduced by [Ti09]. ATL higher-order transformations are classified with respect to usage patterns, but the transformation between different model transformation execution systems is not considered. The work presented here uses, for the first time higher-order transformations to transform a platform-independent model transformation PIM-MT into a platform-specific model transformation PSM-MT @ runtime. Up until now, higher-order transformations and PIM/PSM transformation were considered separately.

The term models@runtime was introduced in a special issue of Computer [BBF09]. The models@runtime workshop 2010 [Be10] presented two application scenarios for automation systems: the reconfiguration of a flexible manufacturing system and the failure detection in industrial automation systems. The reconfiguration of a flexible manufacturing system uses a world model at runtime which is updated according to the state of the production line. The failure detection system uses an engineering knowledge base at runtime to support failure detection of components. Both applications handle models at runtime similar to ACPLT but do not apply model transformation technology nor do they structure the system into PIM to PSM transformations.

This paper proposes the reuse of the platform-specific transformation language ATL as a platform-independent language as it provides a mature tool support and higher-order transformations. Languages such as UMLX [Wi03] or GTXL [La04] have been designed for platform-independent data exchange between transformation tools. Such a language might be an implementation option for the platform-independent language if it were to be supported by a mature development environment.

6 Conclusion and Future Work

The work presented in this paper extends the applicability of current model transformation languages aimed for desktop use to model transformations @ runtime. For this purpose, a new approach was developed, which extends the MDA concept of transformations from platform-independent models (PIM) to platform-specific models (PSM). The new approach transforms platform-independent transformations (PIM-MT)

to platform-specific transformations @ runtime (PSM-MT). The PIM-MT to PSM-MT transformation is a new application of higher-order transformations (HOT). The new concept was verified by an implementation for process control systems, which transforms ATL model transformations by an ATL higher-order transformation to the runtime transformation system ACPLT/RE. For use in the industrial environment, the advantage of this approach is the possibility of being able to transform model transformation definitions to different transformation runtime platforms, e.g. dependent on customer requirements, dependent on a given runtime environment or dependent on system evolution. Furthermore, the ATL platform-independent model transformation (PIM) is easier to test and analyze during the design phase due to its domain-specific syntax and its good tool support. One finding of the work presented here is that the HOT requires an additional transformation in the opposite direction to enter the referenced models of the model transformation specification from the platform-specific representation to the platform-independent specification. In the case of ACPLT/RE and ATL, this includes a transformation from weak typed to strong typed metamodels.

The current implementation of the HOT from ATL to ACPLT/RE is complete with respect to the PSM-MT ACPLT/RE and the current ACPLT/RE application scenarios. ACPLT/RE only requires the implementation of a HOT for the first two packages shown in Figure 7, rule language and system model, and a partial implementation of a HOT for the pattern language. Since the PIM-MT ATL provides more features than ACPLT/RE, a complete HOT from ATL to ACPLT/RE would require the extension of ACPLT/RE by further IEC 61131-3 programming elements. Therefore, the implementation of the HOT is restricted with respect to the PIM-MT ATL, but complete with respect to the PSM-MT ACPLT/RE. The limitations of the current implementation are the reduced ATL language support without inter-rule execution control, without traceability support, and with a reduced set of supported OCL expressions. Therefore, future work will go into two directions: For the platform-specific part, the implementation of traceability and inter-rule execution control will be considered. For the platform-independent part, the use of a model transformation language, which is not as specific as ATL, will be considered to ease the higher-order transformations to a specific implementation.

The platform-independent specification of engineering rules using the reduced ATL language support of the current higher-order transformation implementation is already useful as it avoids many engineering model inconsistencies, and reduces the commissioning time of plant reconfigurations.

7 References

[BBF09] Blair, G.; Bencomo, N.; France, R. B.: Models@run.time. In Computer, 2009; pp. 22–27.

[Be10] Bencomo, N.; Blair, G.; Fleurey, F.; Jeanneret, C. Eds.: Proceedings of the 5th Workshop on Models@run.time at the ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems (MODELS 2010). Oslo, Norway, October 5th, 2010.

[BHW05] Becker, S. M.; Haase, T.; Westfechtel, B.: Model-based a-posteriori integration of engineering tools for incremental development processes. In Software and Systems Modeling, 2005, 4; pp. 123–140.

[In03] International Electrotechnical Commission IEC 61131-3: Programmable controllers – Part 3: Programming languages, 2003.

[Jo06] Jouault, F.: Contribution à l'étude des langages de transformation de modèles, 2006.

[KQE11] Krausser, T.; Quirós, G.; Epple, U.: An IEC-61131-based Rule System for Integrated Automation Engineering: Concept and Case Study: Proceedings of the 9th International Conference on Industrial Informatics (INDIN 2011), 2011; pp. 539–544.

[La04] Lambers, L.: A New Version of GTXL: An Exchange Format for Graph Transformation Systems. In (Tom Mens; Andy Schürr; Gabriele Taentzer Eds.): Proc. Workshop on Graph-Based Tools (GraBaTs'04), Satellite Event of ICGT'04. Elsevier Science, Rom, Italy, 2004; pp. 51-63.

[Le11] ACPLT: ACPLT Technologies. http://www.plt.rwth-aachen.de/en/acplt-technologies/, 15.08.2011.

[LT04] iFBSpro. http://www.ltsoft.de/index.php?id=338, 15.08.2011.

[MM03] MDA Guide Version 1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf, 19.05.2010.

[MV05] Mens, T.; Van Gorp, P.: A Taxonomy of Model Transformation and its Application to Graph Transformation. In (Karsai, G.; Taentzer, G. Eds.): Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005). Tallinn, Estonia, September 28, 2005.

[SE06] Schmitz, S.; Epple, U.: On Rule Based Automation of Automation. In (Troch, I. Eds.): 5th Vienna Symposium on Mathematical Modeling (MATHMOD). ARGESIM ARGE Simulation News Vienna Univ. of Technology, Vienna, 2006.

[Th11] Xtext. http://www.xtext.org, 15.08.2011.

[Ti09] Tisi, M. et al.: On the Use of Higher-Order Model Transformations. In (Paige, R. F.; Hartman, A.; Rensink, A. Eds.): Model driven architecture - foundations and applications. 5th European conference, ECMDA-FA 2009, Enschede, June 23 - 26, 2009, Springer, Berlin, 2009; pp. 18-33

[Wi03] Willink, E. D.: UMLX : A graphical transformation language for MDA. 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture, 2003, Anaheim, 2003.