International Journal of Innovative
Computing, Information and Control
Volume 8, Number 9, September 2012

ICIC International ©2012 ISSN 1349-4198
pp. 6329–6344

A NOVEL APPROACH TO AUTOMATIC CODE GENERATION FOR AUTOMATIC TRAIN PROTECTION

Xiangxian Chen[1], Panfeng Liu[2,*], Xinxi Qiu[2,*], Hai Huang[1] and Huilong Duan[2]

[1]Department of Instrumental Science and Engineering
[2]College of Biomedical Engineering and Instrumentation Science
Zhejiang University
No. 38, Zheda Road, Yuquan Campus, Hangzhou 310027, P. R. China
xxchen99@gmail.com; {hhai; duanhl}@zju.edu.cn
*Corresponding authors: lylpf0122@163.com; qiuxinxi@zdwxjd.com

Received June 2011; revised November 2011

Abstract. *The Model-Driven Development (MDD) method has been proposed to solve the problem of high error rates in the conventional manual implementation of software. This paper focuses on the modeling of a common framework for an Automatic Train Protection (ATP) system with an MDD method, which includes interface, architecture and function models. The purpose of this approach, which combines the source file and template, is to perform automatic code generation. The source file is generated based on the model that is established using the MDD method; thus, its logic is guaranteed to be correct and the template is customized according to the coding style of the model. A toolkit to automatically generate code for an ATP system is also developed, which allows the entire development process to be automated.*

Keywords: Automatic train protection, Model-driven develop, State machine, Code generation, XML, XSLT

1. **Introduction.** In recent years, urban rail transit develops rapidly; however, the technologies of signaling systems of this field in China are far behind the developed countries. The Automatic Train Protection (ATP) system is a key system for ensuring the safety and high-efficiency of the urban rail transit. The safety of such software is critical for us, especially when failures may lead to catastrophes where people die or economics are lost. Therefore, we propose an automatic development method to ensure the safety.

The ATP system is characterized by the extensive use of control mode logic and message analysis algorithms. The key issue of the development is to guarantee the safety by verifying correctness of ATP logic. Additionally, it is also a main concern to control the development costs and promote the efficiency. Thus, many researchers have contributed to these. Some studies concentrate on using model-based tools [1] with safety verification to ensure high quality development in ATP systems. Currently, the Unified Modeling Language (UML) based model approach [2] is commonly used for development in safety-critical fields, but because the requirements change continuously, it is hard to ensure synchronization between the UML models and the implementation. Other studies concentrate on applying formal methods such as B Method [3], state flow [4] or finite state machine [5] to verify the correctness of ATP logic. However, formal methods are difficult to master because of the complex semantics. Recently, there are also some researchers devoted to automation development methods [6-10]. These approaches not only avoid coding errors but also substantially shorten the development time. There are already some commercial software packages applied in some safety-related areas. These methods

are more rigorous in semantics or automatic than traditional develop methods, but they cannot guarantee the logic to be correct. Moreover, the commercial software can only generate code with a fixed format, and it is rather expensive. For these reasons, it has not been widely used in the development of ATP systems.

However, an approach which is able to not only increase developing efficiency but also guarantee correctness of ATP logic is still lacked. Based on the previous work, an automatic code generation approach based on the Model-Driven Development (MDD) is proposed to solve these problems. This innovation approach combines the models and the implementation, and enables developers to focus on model design instead on programming. As long as the models are correct, the generated code will be accurate and precise. In this way, the development efficiency is also increased by intergradations directly from the model and automatically codes generating.

In this method, a common framework model for an ATP system is established according to the function features, which includes interface models, architecture model and function models. Then, the Extensible Markup Language (XML) model files can be generated from these models. Combining the model files and code template customized according to the coding style or specification, the target source codes can be generated automatically. Also, a toolkit is developed to implement the processes of this method proposed in this paper. The practical results based on the data of Shenyang metro line 1 show that the source codes of ATP software generated by the toolkit well satisfied the needs of urban rail transit system.

2. Design of an ATP Model.

According to the current definition of a Communication Based Train Control (CBTC) system [11], a deployment diagram of a Carborne Controller (CC) subsystem of a CBTC system is shown in Figure 1(a). The CC system, which is composed of an Automatic Train Operation (ATO) and ATP system, has the responsibility of determining the position of the train, monitoring train speed, assuring appropriate braking sequences, managing the control mode, and responsible for safe movement of the train within Movement Authority Limit (MAL) provided by the Zone Controller (ZC) [11,12].

2.1. Main functions of the ATP system.

As the key subsystem of the CC system, the external interface of the ATP system is shown in Figure 1(b). The ATP system calculates the train speed with the Optical Pulse Generator (OPG), obtains the train position via a Transponder Interrogator Antenna (TIA) by reading the trackside beacon, and controls the vehicle signal devices via the Rolling Stock (RS) interface. The ATO system and CC on the other end of the train (CC2) are interfaced to achieve speed regulation, stopping at stations, door control, brake control and automatic turn-back functions.

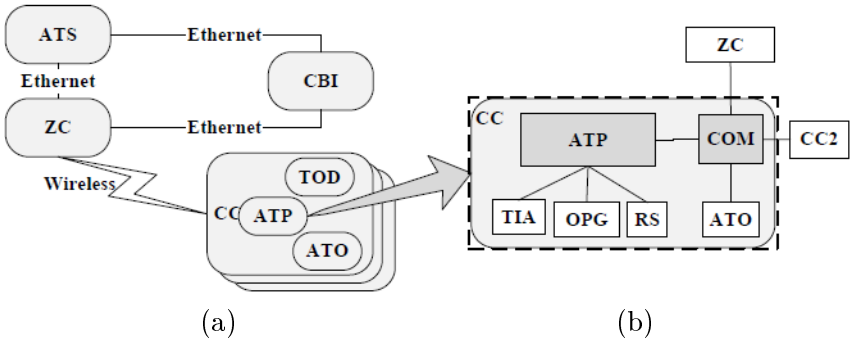


FIGURE 1. Deployment diagram of the ATP external interface

2.2. **Challenges of ATP development.** The Safety Integrity Level of an ATP system is SIL4 in the CENELEC 50128 standard [13], which gives a reasonable and effective development method. This standard demands that system's safety is ensured from the development process, including the safety development life cycle of the V-model and the use of a formal modeling language. In the development of ATP software systems [14,15], there are many challenges to face including the following:

- The control of system complexity and scale;
- The consistency of software requirements and source code;
- The avoidance of as many artificial coding errors as possible;
- The implementation and updating of code in an efficient way.

These challenges are especially critical in the current development of safety-related systems. So, the Model-driven development approach is proposed to solve these problems.

2.3. **Composition of an ATP model.** The system requirements specification is composed of system interface requirements and functional requirements, which include a description of the module input, information regarding output data, and details of the system's functionality. The features of ATP systems from various major manufacturers are compared in Table 1. According to these characteristics, the ATP model is established, and it's composed of three parts: the interface, architecture and functional models, as shown in Figure 2.

- According to the interface requirements, the interface model is designed to describe the interfaces to the external systems, including interface models of the rolling stock signal system, speedometer device, beacon interrogator, and communication system.
- According to the ATP system functional requirements, the ATP functional models are designed by a data flow diagram model or a state machine model, which describe the input and output variables and the internal details of the functional unit.
- According to the correlations between the ATP functions, the architectural model is designed to describe the data interface and the correlations between the functions, which comprise the functional models and the interaction connections.

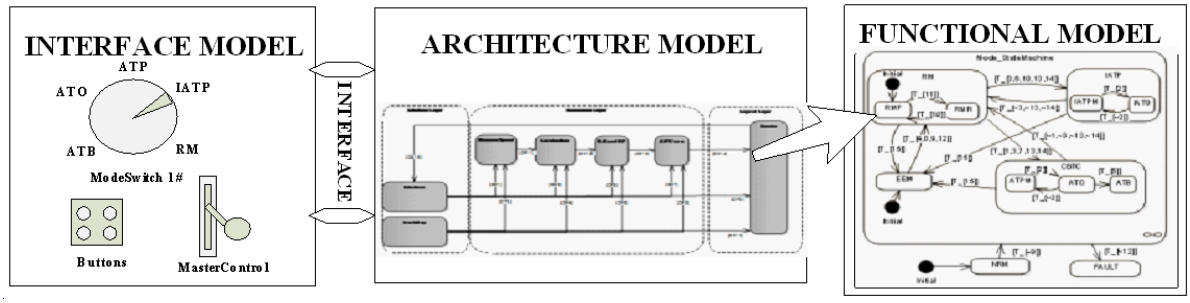


FIGURE 2. Structure of an ATP model

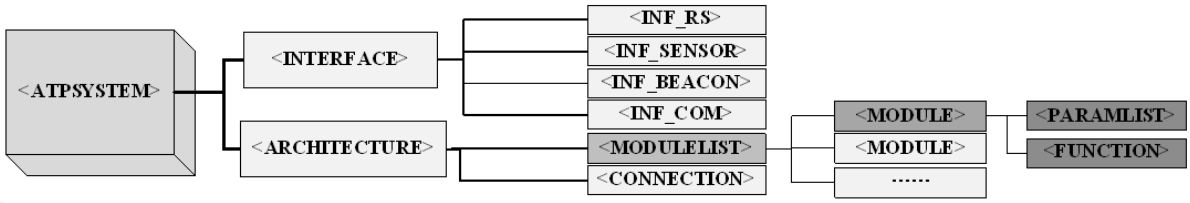


FIGURE 3. Data structure of an ATP model

TABLE 1. ATP systems of major manufacturers

Manufacturers	Main Function of ATP system	Interface of ATP system
SIEMENS (ATC)	*The ATP system is responsible for the vital operation of the trains. It has different tasks to ensure the safety in a metro system. The ATP continuously detects the position and the speed of the train, supervises the speed restrictions, controls the train doors, supervises the platform screen, tracks all equipped trains, respects the interlocking conditions such as target points position or supervision.*	*The subsystems connected to the ATP on-board computer unit are the balise antenna, the odometer (OPG) and the radar unit. The radio has got a receiving and a transmitting antenna. Other connections are established to the ATO and the driver's HMI.*
ALSTOM (URBALISTM)	*The ATP system is to achieve the following functions: The ATP provides the fault-safety train location function; besides it ensures the emergency brake to prevent the train crossed the protection point, over-speed or other safety-related conditions. It also provides authority of doors, and status monitoring, such as alarm, the driving mode, door status, link status, departure instructions.*	*The position information is collected through the vehicle odometer and beacon read by antenna installed on the line. The ATP receives railway track line information and real-time track status continuously, and feeds back the location information to ZC.*
ALCATEL (SelTrac®)	*The ATP is responsible for safety functions the of ATC system. The ATP's basic responsibility is as follows: prevents train collisions caused by hostile running; prevents dangerous incidents to passengers due to unexpected door open, or train slipping; prevents the train running over the speed limit or recommend speed to cause damage dangerous to the train.*	*The ATP system interfaced to the following equipment: driver display unit, beacon interrogator antenna, accelerometer, speed sensor, vehicle DCS and wireless unit.*
USSI	*The ATP functions assure the safety of train operations. All ATP functions are implemented in accordance with the failsafe principle. The Carborne Subsystem has the responsibility of determining the position of the train, monitoring train speed, assuring appropriate braking sequences as necessary, managing the control mode of the train, and controlling the train according to the information provided by ZC.*	*The ATP system interfaces to speed sensors, accelerometer and the transponder interrogator to determine the position of the train. A Train Operator Display is interfaced to the ATP to display driving information, equipment status, and alarms to the operator.*

The data structure is shown in Figure 3: the root node <ATPSYSTEM> consists of an interface model node <INTERFACE> and an architecture model node <ARCHITECTURE>. The architecture node <ARCHITECTURE> not only defines the structure type, but also determines the division and the correlations of the functional models. The functional model node <MODULE> includes multiple subordinate nodes <FUNCTION> to complete a full description of the functions.

2.4. **Model elements.** In this paper, two basic types' elements are designed for description of the functions: Nodes and Connections.

- Element Node: refers to entity object of the model, such as function node or module node;

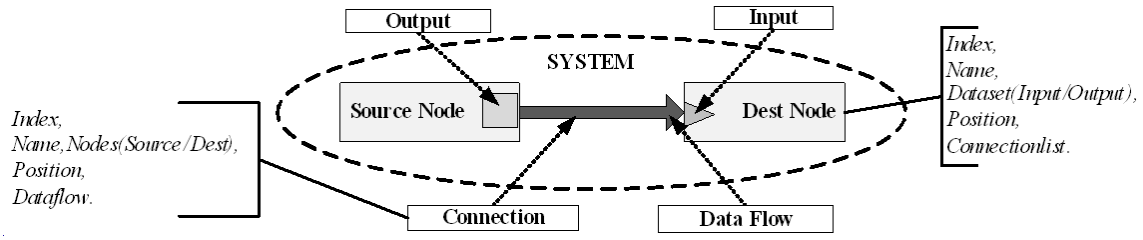


FIGURE 4. An example of the elements

- Element Connection: refers to the relationship between the objects, such as the data exchange of the modules. The properties are shown in Figure 4.

There are also certain rules and constraints in or among these elements. For example, as shown in Figure 4, the constraints between the nodes and the corresponding connect are:

- Data flow of the Connection must be the subset of the output data from source node;
- Data flow of the Connection must be the subset of the input data from destination node; Or else, this connection is invalid for this model.

So the core of the model structure for ATP software system is the objects are expressed in format of nodes, the relationships in format of connections, and the configurable module structure in format of properties, characteristics and rules. The models we discuss in the following chapters are all based on these elements and constraints.

3. **Interface Model.** According to the applications environment and the system architecture, we design the interface models, including an RS interface model, an OPG interface model, a TIA interface model and a COM interface model (as shown in Figure 5). In these models, the commonly used interfaces and logic are pre-defined and it is easily to customize by users to generate required interface models. Moreover, the data obtained from these interfaces are used throughout the entire modeling process.

3.1. **Rolling stock model.** The interface between the ATP system and the rolling stock comprises discrete input/output (IO) signals, including vital input (VI)/non-vital input (NVI) and the vital output (VO)/non-vital output (NVO). All of the IO signals designed in this rolling stock interface model are listed in Table 2.

Each input/output signal has similar properties, as listed below:

- T: type of the signal (VI/NVI/VO/NVO);
- V: signal value;
- B: the input/output board position of the signal connection pins (ID and bits);

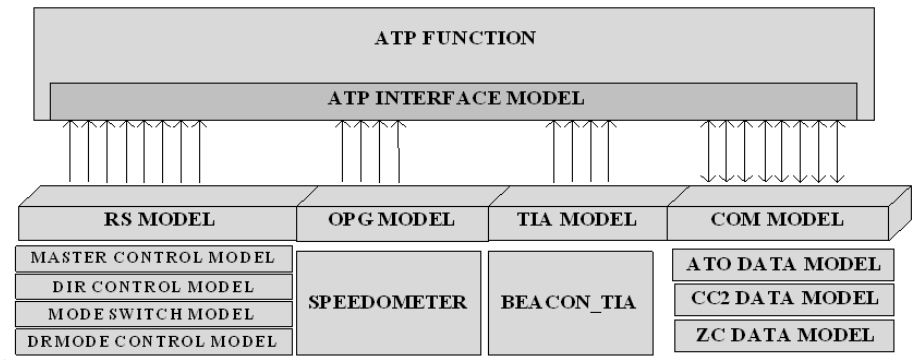


FIGURE 5. ATP interface model

TABLE 2. IO signals in the rolling stock interface model

NVI_Depart	*NVI_EBA*	*NVI_SBA*	*NVI_ADOADC*	*NVI_ADOMDC*	*NVI_MDOMDC*
VI_KeyCC1	*VI_KeyCC2*	*VI_MS1_RM*	*VI_MS1_IATP*	*VI_MS1_ATPM*	*VI_MS1_ATO*
VI_MS1_ATB	*VI_MS2*	*VI_DirCtrl_FWD*	*VI_DirCtrl_REV*	*VI_DirCtrl_NEUT*	*VI_MC_Coast*
VI_MC_FSB	*VI_MC_FB*	*VI_DoorNOR*	*VI_ADC*	*VI_IATP_Release*	*VI_ATB*
VI_Integrity	*NVO_OLD*	*NVO_ORD*	*NVO_CLD*	*NVO_CRD*	
VO_EBR_P	*VO_EBR_N*	*VO_RDE*	*VO_LDE*	*VO_ATB_Key*	*VO_PE*

TABLE 3. RULES of the RS interface model

RULES	DESCRIPTION
RULE_INF_RS_1	*SINGLE: only one signal is valid;*
RULE_INF_RS_2	*NONEORSINGLE: one signal or no signal is valid;*
RULE_INF_RS_3	*ALL: all signals must be valid.*

- D: the minimum duration of the valid signal;
- R: some interrelated constraints, such as mutually exclusive exits among the input/output signals. Depending on the various constraint correlations among the signals, there are three types of rules in this RS interface model (Table 3).

Based on the properties that are described above and the constraints between the signals, the value of an IO signal can be expressed by the following formula: $V = IO(R, T, B, D)$.

3.2. **Speedometer device model.** Typically, an ATP system uses a speedometer to measure the train speed. A speedometer generates N pulses in each circle of the wheel, and the frequency of the pulse is proportional to the angular velocity. According to the pulse count and the pulse frequency, the train speed, travel distance and driving direction can be determined. Therefore, the properties of a speedometer are as follows:

- PD: Pulse distance;
- S: Increase/decrease the sign of the pulse count (UP/DOWN);
- F: Working flag of the speedometer (OK/ERROR);
- PC: Cumulative count of the pulse;
- PF: Frequency of the pulse.

Based on the actual operation of the train speed per cycle, the speedometer model can be expressed with the following formula: $NVSpd = Speed(PD, S, F, PC, PF)$.

Depending on the above description, the calculation rules of the pulse count per cycle and the original train speed are as follows:

TABLE 4. RULES of the speedometer interface model

RULES	DESCRIPTION
RULE_INF_SENSOR_1	$GETPULSEPERCYC : PULSEPERCYC = (PULSECOUNT - PREPULSECOUNT) * SIGN$
RULE_INF_SENSOR_2	$GETRAWSPEED : NVSPEED = PULSEFREQ * PULSEDIST$

3.3. **Beacon interrogator model.** When the train sweeps over the beacon, the ATP system reads the information of beacon through the transponder interrogator antenna installed under the train. The system obtains the location of the beacon by the information and accomplishes a relocation or wheel diameter calibration functions. Thus, the main properties of the beacon interrogator are as follows:

- S: the status of the interrogator host (OK/ERROR);
- D: the distance from the interrogator to the train's front-end;
- F: flag of "is beacon detected" (TRUE/FALSE);
- ID: ID of "detected beacon read";
- T: type of beacon (USA/EU);
- P: position of beacon in the track map database;
- FPC: frozen pulse count when the beacon is read.

In accordance with the interrogating result of the transponder interrogator antenna, the model of beacon TIA is expressed as follows: $(ID, P) = BcnTIA(S, F, ID, P, FPC)$.

When the status of an interrogator host is OK with the beacon read, the beacon is valid. The train's location can be relocated when it reads the beacon because the train's location is calculated from the beacon position and the distance from the interrogator to the front-end of the train. Therefore, the rules are as follows:

TABLE 5. RULES of the beacon TIA interface model

RULES	DESCRIPTION
RULE_INF_BEACON_1	$CHECKBCN : BCNID = (STATUS\&\&READBCN) * BCNID$
RULE_INF_BEACON_2	$GETRELOCPOS : RSPOS = BCNPOS + TIA2HEADDIST$

3.4. **Communication model.** The ATP system communicates with the ATO and CC2 system every cycle and with the ZC after the train locates. The ATP system applies to establish or disconnect with the communication target per cycle, according to the current operation of the train, and checks the connection status of each target per cycle. Once the connection is successful, the ATP will receive or send the communication message; meanwhile, if no packet is received within the TIMEOUT time, it is considered to be a timeout event with completion of a certain appropriate logic process. Thus, communication targets usually have the following properties:

- ID: ID of communication destination (ATO/CC2/ZC);
- LS: link status of each communication target (OK/ERROR);
- LR: link request of each communication target (TRUE/FALSE);
- T: time-out of effective communication message;
- D: type of dataflow (INPUT/OUTPUT).

Thus, the model can be expressed with the following formula: $D = COM(ID, LS, LR, T)$.

When the status of a connection is OK, and it is not a timeout, the dataflow is valid for this communication target. This rule is expressed in Table 6.

4. **Architecture Model.** Considering the characteristics of the network topology, software systems can often be designed to sequence structure, network structure, hierarchical structure or star-shaped structure. With regard to the complexity of the logic and functions, this article provides a design for the hierarchical and star-shaped structure architecture model for ATP.

TABLE 6. RULES of a communication interface model

RULES	DESCRIPTION
RULE_INF_COM_1	$CHECKMESSAGE : ATAITEMS = (LINKSTATUS\&\&(!TIMEOUT)) * DATAITEMS$

4.1. **Description of the architecture model.** The architecture model provides a description for the input and output information of the modules and the interaction connections between modules. Therefore, the architecture model has the following properties:

- *TYPE*: Type of architecture model (hierarchical/star-shaped);
- *MODULELIST*: list of modules (input/output);
- *CONNETCTION*: corrections between modules (source/destination and data flow).

```
<ARCHITECTURE TYPE= "StarModel">
<MODULELIST>
<MODULE Name = "Interface">
    <PARAMLIST>
    <PARAM Name= "SENSOR_STATUS">
    <SCOPE>public</SCOPE>
    <DATATYPE>BOOL</DATATYPE>
    </PARAM>
    </PARAMLIST>
    <FUNCTION></FUNCTION>
</MODULE>
</MODULELIST>
<CONNECTIONLIST>
<CONNECTION>
<SOURCE>MeasureSpeed</SOURCE>
<DESTNATION>Localization</DESTNATION>
<DATAFLOW>
        <DATA></DATA>
        </DATAFLOW>
</CONNECTION>
<CONNECTION></CONNECTION>
</CONNECTIONLIST>
</ARCHITECTURE>
```

FIGURE 6. Data structure of architecture model

For the data structure of the architecture model described in Figure 6, the architecture model node <ARCHITECTURE> defines the type of structural model to be a hierarchical structure or a star-shaped structure, which also determines the division of the functional model nodes <MODULE> and the data flow. The <MODULE> node describes the input and output parameters. The module connection list node <CONNECTIONLIST> includes multiple lists of <CONNECTION> nodes that achieve a complete description of the interaction connections. The data flow node <DATAFLOW> is composed of dozens of data item nodes <DATA>. All of these data items are derived from global input and output variables of <MODULE> with the same data type.

4.2. **Hierarchical model.** The hierarchical model has the advantages of distinct gradation: each layer is concerned only with the current content, which shields the implementation details of the other layers. Also, each level provides services to the upper layer, and requests services from the lower layer. So, it is consistent with the actual characteristics of ATP functions. The hierarchical architecture is shown in Figure 7(a), which shows that the seven models can be divided into three layers: the interface layer, business layer and logic layer.

The bottom interface layer realizes the platform management functions and mainly addresses the preprocessing of the input interface data and the final processing of the output data. The business layer implements data conversion and accomplishes the calculation of the main information, meanwhile providing data for the logic layer. The top layer accomplishes the collection and management of all of the safety logic. It is the ultimate execution unit of the ATP safety logic, and is responsible for managing the train mode and safety monitoring system. All the logic outputs follow fail-safe principles.

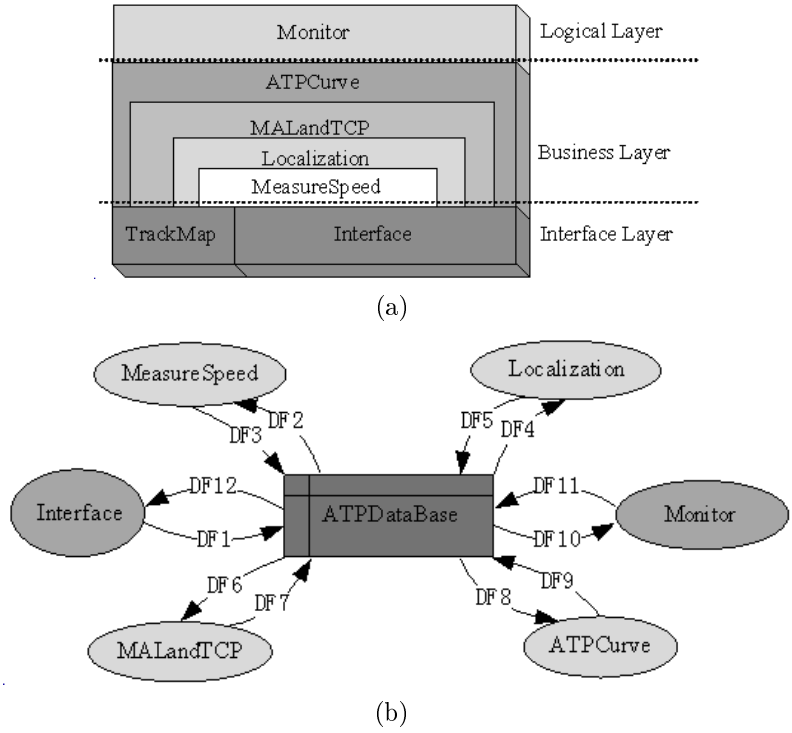


FIGURE 7. ATP architecture model

4.3. **Star-shaped model.** The hierarchical model has the advantages of distinct gradation and high flexibility; however, the coupling between modules is tight. However, a star-shaped model can be used to meet the demands of high independence. In the star-shaped architecture model (shown in Figure 7(b)), a data sharing module *ATPDataBase* is designed to cover all the interactive interfaces, and it provides a good solution for ATP to solve the complex logic data interaction. The data interfaces of this module are categorized in Table 7.

There are two types of data flow: updating or getting information (UPDATE/GET). According to the method described in Section 4.1, we can determine the main data flow between the modules (Figure 7). The relationships between the *ATPDataBase* interface types (DB) and the data flow between modules (DF) are shown in Table 8. For example, in Figure 7, the data flow *DF6* means obtaining information from the *ATPDataBase* and providing it to the *MALandTCP* module; we need three types of *ATPDataBase* interfaces (*DB2, DB4, DB6*) to obtain the interface-related, speed-related and location-related data.

The model of the ATP star-shaped architecture not only reduces the coupling between the module data but also improves system safety with defensive measures in the *ATPDataBase* module, but it results in data redundancy. Consequently, users should select the appropriate model to build the ATP architecture model according to their actual needs.

4.4. **Functional model and description.** After determining the system's interface model, architecture model and the data flow between the modules, we use the approach of data flow diagram [16] and state machine [17] to model the detailed implementation within each function module. In the functional models, there are a number of data processing units included, which are abstracted as an operator meta-models. Each meta-model includes the type of operator, two inputs and one output, which is expressed as the following formula: $Result = Oper[Operand1, Operand2]$.

TABLE 7. Data flow interfaces of the ATPDataBase module

ID	DESCRIPTION
DB1	*Updates the interface data and is responsible for updating the data that goes to the data sharing module.*
DB2	*Receives the interface data and is responsible for providing the interface-related data to the target module.*
DB3	*Updates the speed data and is responsible for updating the speed data that goes to the data sharing module.*
DB4	*Obtains the speed data and is responsible for providing the speed-related data to the target module.*
DB5	*Updates the location data and is responsible for providing the updated location data to the data sharing module.*
DB6	*Obtains the location-related data and is responsible for providing the location-related data to the target module.*
DB7	*Updates MAL and target point information and is responsible for providing the MAL and target point to the data sharing module.*
DB8	*Obtains the MAL and target point information and is responsible for providing the MAL and target point to the target module.*
DB9	*Updates the ATP curve data and is responsible for providing the updated curve data to the data sharing module.*
DB10	*Obtains the ATP curve data and is responsible for providing the curve data to the target module.*
DB11	*Updates the train monitoring data and is responsible for providing the updated train monitoring data to the data sharing module.*
DB12	*Obtains the train monitoring data and is responsible for providing the train monitoring relevant data to the target module.*

TABLE 8. Relationship mapping of DB and DF

DATA FLOW	IDENTIFIER OF INF TYPE	DATA FLOW	IDENTIFIER OF INF TYPE
DF1	*DB1*	**DF7**	*DB7*
DF2	*DB2*	**DF8**	*DB2, DB4, DB6, DB8*
DF3	*DB3*	**DF9**	*DB9*
DF4	*DB2, DB4*	**DF10**	*DB2, DB4, DB6, DB8, DB10*
DF5	*DB5*	**DF11**	*DB11*
DF6	*DB2, DB4, DB6*	**DF12**	*DB2, DB4, DB6, DB8, DB10, DB12*

A function operating model is composed of multiple operator meta-models. The data flow between the operators or functions constitutes the ATP logic functions. The operator meta-models or function models involved in this paper are shown in Table 9. Consequently, the function is described in precise mathematical semantics, which is accurate, consistent and unambiguous with respect to the designed model.

5. **Code Generation from Model.** Based on the models built in the above chapters, a series of model files will be generated. We choose XML to describe the models, because of its flexibility and scalability. Besides, XML and XSLT provide a good programming model of metadata, and XSLT can convert XML files into any format [19,20]. So, we propose the "XML + XSLT" method to implement code generation in this paper.

TABLE 9. The operation models and function models

OPERATION	EXPRESSION	DESCRIPTION	EXPRESSION	DESCRIPTION
ARITHMETIC OPERATION	$a+b$	$Add[a,b]$	$a \times b$	$Mult[a,b]$
	$a-b$	$Sub[a,b]$	a/b	$Div[a,b]$
	$a\%b$	$Mod[a,b]$		
LOGIC OPERATION	$a\&\&b$	$And[a,b]$	$a\&b$	$BitAnd[a,b]$
	$a\|\|b$	$Or[a,b]$	$a\|b$	$BitOr[a,b]$
	$!a$	$Not[a]$	$a>b$	$GreaterThan[a,b]$
	$a \ll b$	$Shiftleft[a,b]$	$a<b$	$LessThan[a,b]$
	$a \gg b$	$Shiftright[a,b]$	$a=b$	$Equal[a,b]$
FUNCTION OPERATION	$AVG[a,b]$	$Sub[Add[a,b],2]$	$SUM[a,b,c]$	$Add[Add[a,b],c]$
	$User\_defined$			

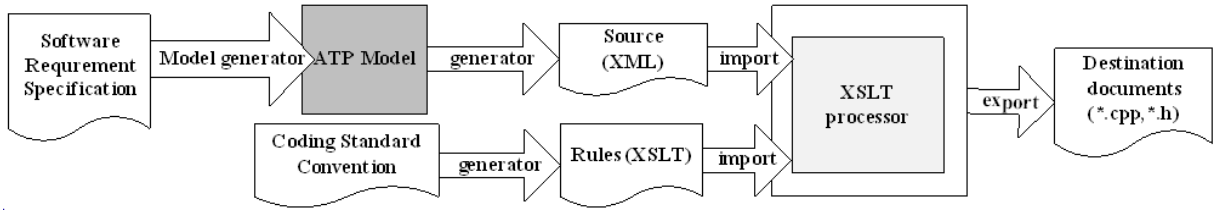


FIGURE 8. Schematic diagram of code to generate an implementation

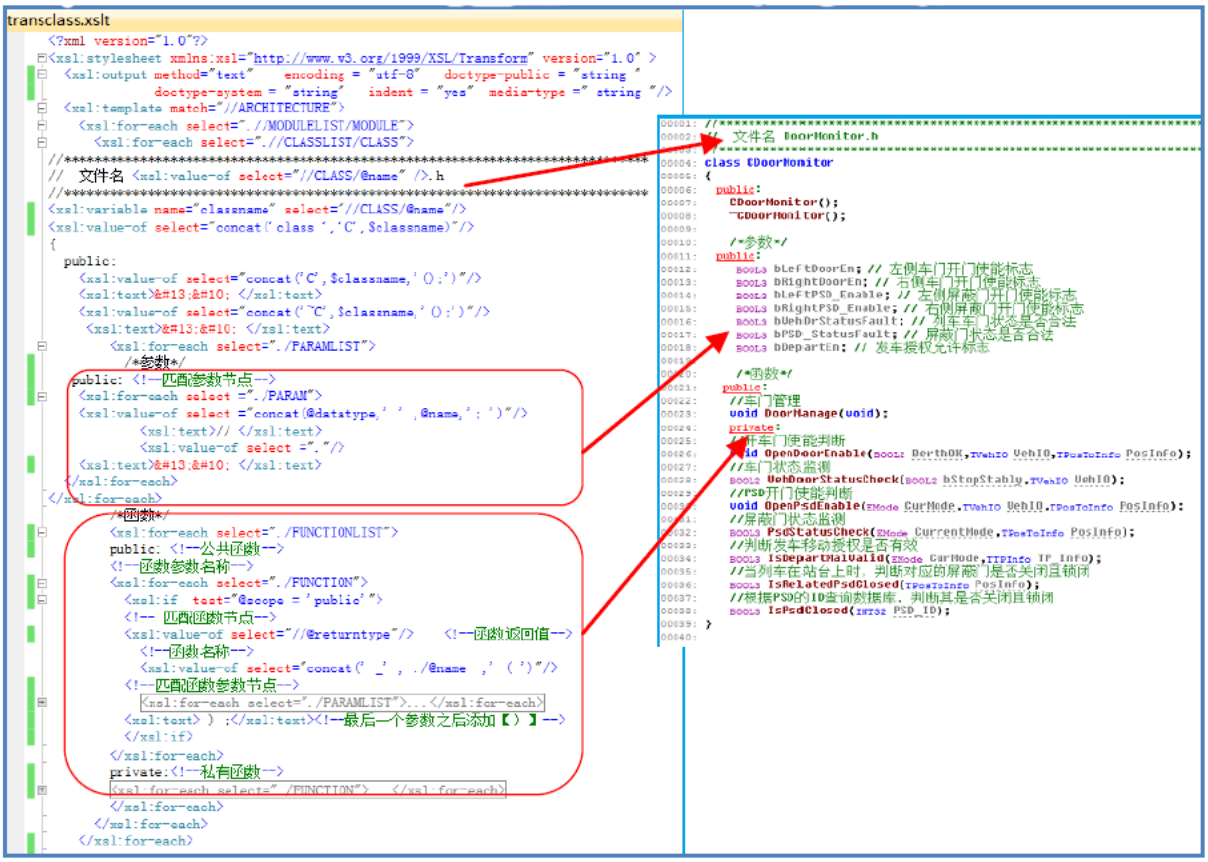


FIGURE 9. An example of code generation with XSLT

The code generation schematic diagram of the ATP model is shown in Figure 8. The XML source file is generated according to the data structure of the model, which is

designed in Sections 3 and 4. Based on the coding specification and documentation conventions [21], the XSLT template file customized by the user is generated. With the use of a conversion tool (XSLT processor), we can automatically generate C-language target source code.

Figure 9 shows an example of code generation with XSLT. The left file "transclass.xslt" is the template file which translates the model file shown in Figure 8. By loaded into the XSLT processor with the XML files shown in Figure 8, finally the processor generates the target codes as shown in the right of Figure 9. In this way, the naming and coding style can be consistent throughout all the progresses, greatly improving the development quality. When updated, by changing the source model files and re-running the generator, the code will be renovated, so we can accomplish "long term" responsibility for renewed code without changing the generator. Furthermore, it will promote the development efficiency by using the model files and code template as a prototype for the development of a similar system.

6. **Implementation and Case Studies.** According to the approach described above, an automatic code generation toolkit is implemented. The toolkit consists of three software tools: the ATP model generator, the Code template generator and the Auto-code generator. First, the detailed models are established from the configurable ATP framework model; then, the template XSLT files are obtained from the coding specification; finally, the final codes are generated based on the models and the templates, as shown in Figure 9.

6.1. **ATP model generator.** This toolkit builds the interface model, architecture model and functional models for ATP system, as is shown in Figure 10.

The interface model includes RS interface, SENSOR interface, BEACON interface and COM interface model. Users can directly use this model or customize the model according to actual demands. An example of setting IO interface model in shown in Figure 11: A

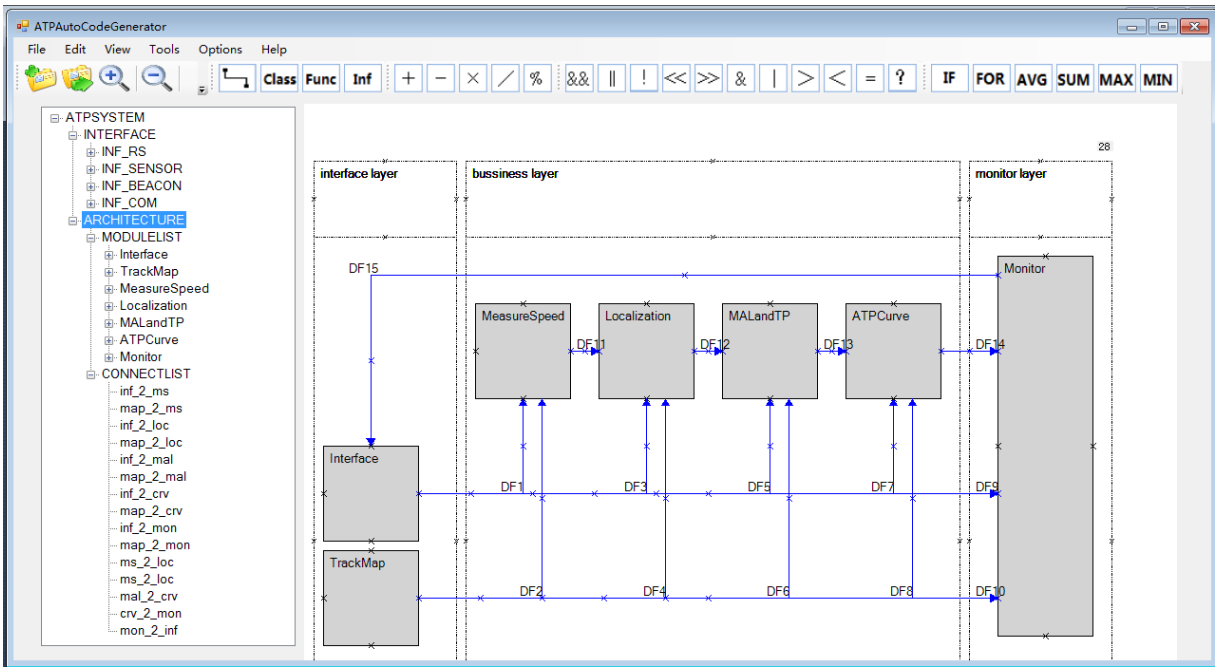
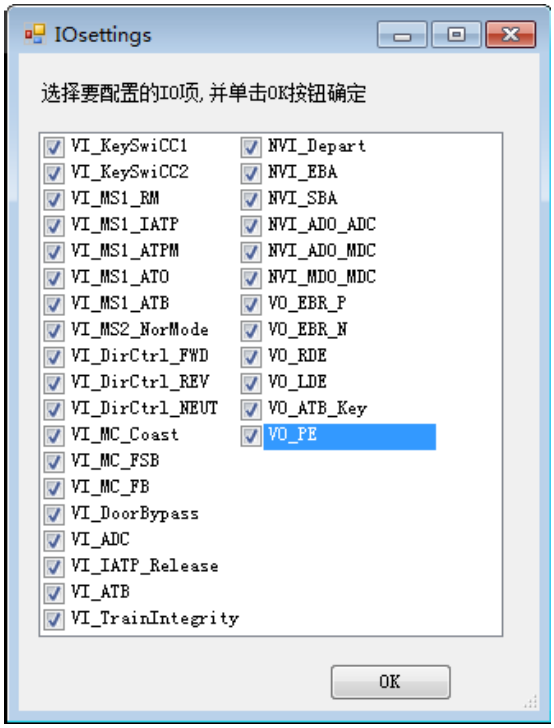
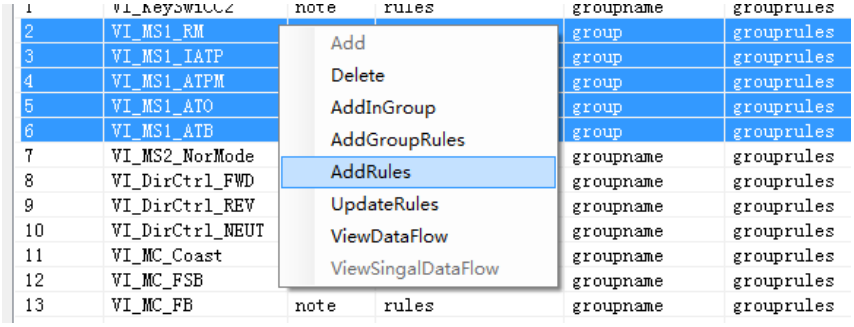


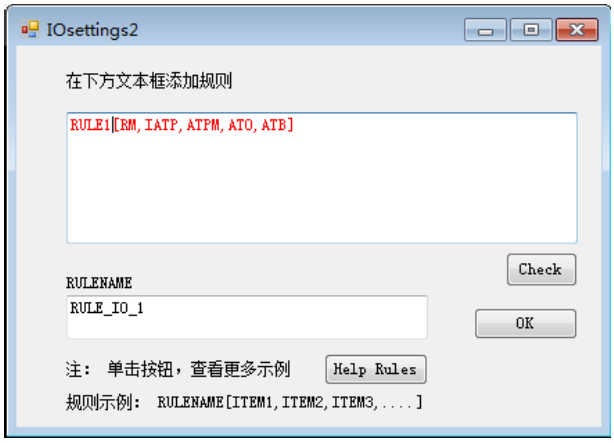
FIGURE 10. Interface of an ATP model generator



(a) IO configuration

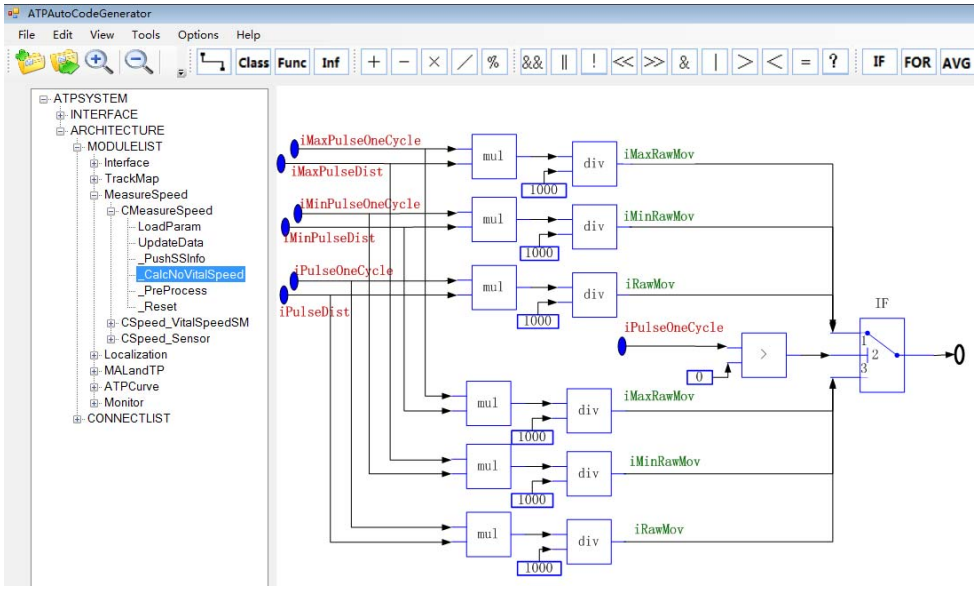


(b) Add rules for IO

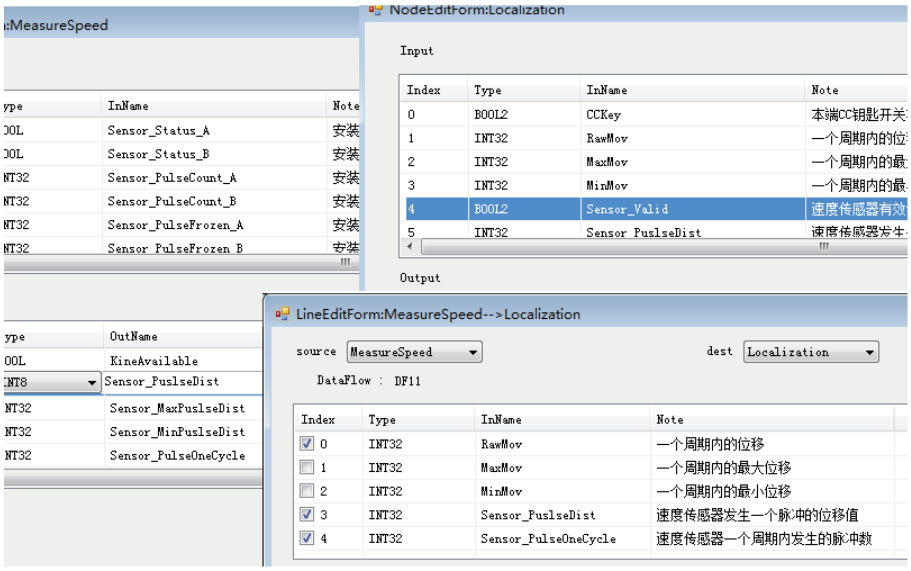


(c) Rules edit

FIGURE 11. Settings of INF_RS interface



(a)



(b)

FIGURE 12. (a) Data flow of function [*_CalcNoVitalSpeed*], (b) an example of constraints between the modules

list of input/output RS models could be chose in Figure 11(a); developers can configure rules for each IO signal, or a series of signals, as shown in Figure 11(b) and Figure 11(c).

According to the architecture model that the user selects, different partition of modules will be generated; after that, developers can design each functional module through an operator meta-model or a function model via this toolkit. An example of function model from [*MeasureSpeed*] module is shown in Figure 12(a). Meanwhile, as described in Section 2.4, the constraints are also taken into account in this toolkit. For example, the dataflow [DF11] in Figure 10 comes from Module [*MeasureSpeed*] to Module [*Localizaiton*], and the parameters of the two modules are shown in Figure 12(b), so the dataflow of this connection must be subset of their intersection.

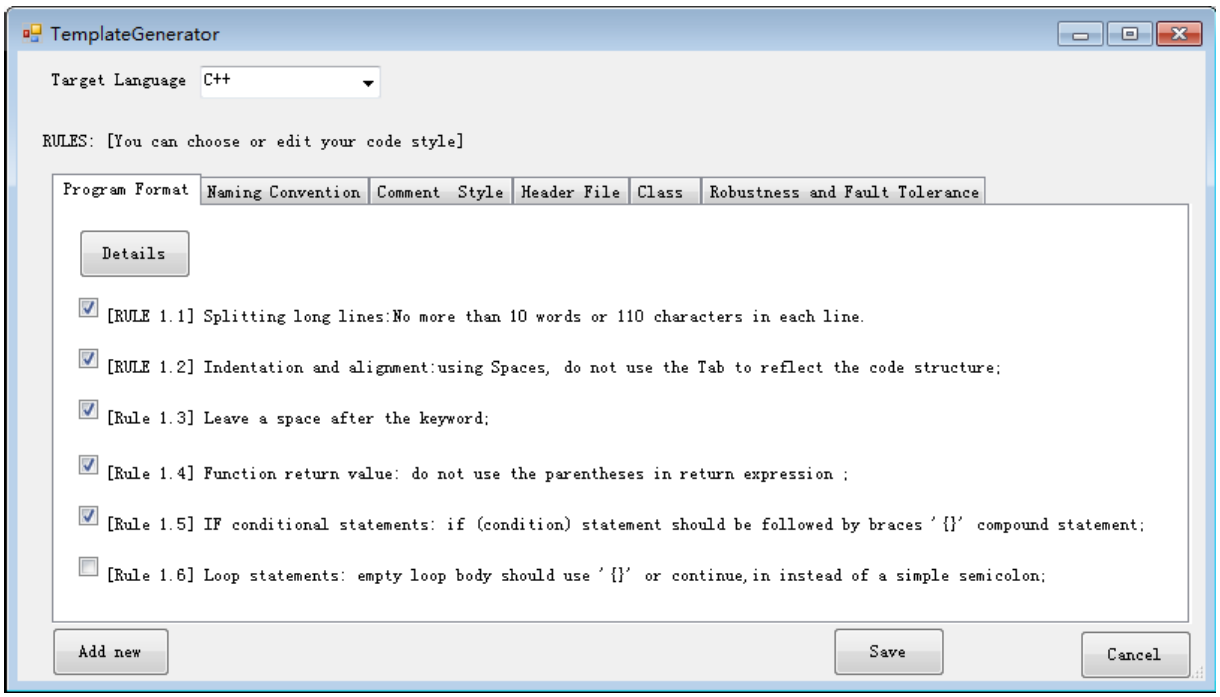


FIGURE 13. Interface of a code template generator

6.2. **Code template generator.** As shown in Figure 13, it generates the style rules of the source code according to the coding specification of the project. Developers can edit and configure the code style via this toolkit.

6.3. **Auto-code generator.** After the above steps, based on the template and model files, the Auto-Code generator automatically generates C language code as shown in Figure 9.

7. **Conclusions.** In safety critical areas, a high degree of safety and reliability are key factors for the system. Model-driven techniques help to avoid errors in the software development process. This paper presented a novel approach to model-driven code generation method for the ATP system, which built a model of a common ATP framework including interface, architecture, and function models. Developers can customize these models according to their actual needs. A set of toolkits is also developed to implement this procedure automatically. According to the XML model files that are generated from the models by the toolkits, together with the XSLT code template files, the target codes generated in an automatic way. A significant decrease in development time, number of errors was observed by using this tool, while consistency between requirements and source code is guaranteed. The automatic code generation approach proposed in this paper not only ensures the logical correctness of the ATP, but also provides a convenient and effective path for development. The ATP codes generated by the approach proposed in this paper are emulated on a simulation platform, and the simulation results are well satisfied the design objective of the system.

Acknowledgment. This work is partially supported by National Science and Technology Infrastructure Program of China (Grant No.: 2011BAG01B03). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] H. Wang, C. Gao and S. Liu, Model-based software development for automatic train protection system, *Proc. of Computational Intelligence and Industrial Applications*, pp.463-466, 2009.

[2] T. Lodderstedt, D. Basin and J. Doser, SecureUML a UML-based modeling language for model-driven security, *Proc. of Unified Modeling Language*, pp.426-441, 2002.

[3] F. Badeau and A. Amelot, Using B as a high level programming language in an industrial project, *Proc. of Formal Specification and Development in Z and B*, pp.334-354, 2005.

[4] A. E. Haxthausen, *An Introduction to Formal Methods for the Development of Safety-Critical Applications*, http://www2.imm.dtu.dk/courses/02263/F11/Files/FormalMethodsNoteTS.pdf.

[5] F. Lindlar and A. Zimmermann, A code generation tool for embedded automotive systems based on finite state machines, *Proc. of Industrial Informatics*, pp.1539-1544, 2008.

[6] B. Vogel-Heuser, D. Witsch and U. Katzke, Automatic code generation from a UML model to JEC 61131-3 and system configuration tools, *Proc. of Control and Automation*, pp.1034-1039, 2005.

[7] E. Denney and S. Trac, A software safety certification tool for automatically generated guidance, navigation and control code, *Proc. of Aerospace Conference*, pp.1-11, 2008.

[8] A. Ferrari, A. Fantechi, S. Bacherini and N. Zingoni, Modeling guidelines for code generation in the railway signaling context, *Proc. of the 1st NASA Formal Methods Symposium*, pp.166-170, 2009.

[9] A. Ferrari, M. Papini, A. Fantechi and D. Grasso, An industrial application of formal model based development the MetrO Rio ATP case, *Proc. of the 2nd International Workshop on Software Engineering for Resilient Systems*, pp.71-76, 2010.

[10] B. Selic, The pragmatics of model-driven development, *IEEE Software*, pp.19-25, 2003.

[11] IEEE standard for communications-based train control (CBTC) performance and functional requirements, *IEEE STD 1474.1*, 1999.

[12] IEEE recommended practice for communications-based train control (CBTC) system design and functional allocations, *IEEE STD 1474.3*, 2008.

[13] CENELEC, Railway applications – Communications, signaling and processing systems – Software for railway control and protection systems, *European Committee for Electrotechnical EN 50128*, 2001.

[14] L. Yang and K. Li, The railway transportation planning problem and its genetic algorithm based tabu search algorithm, *ICIC Express Letters*, vol.3, no.3(A), pp.361-366, 2009.

[15] G. Vachkov, Growing neural models for process identification and data analysis, *International Journal of Innovative Computing, Information and Control*, vol.2, no.1, pp.101-123, 2006.

[16] E. Gamma, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1995.

[17] F. Wagner, R. Schmuki, T. Wagner and P. Wolstenholme, *Modeling Software with Finite State Machines: A Practical Approach*, Auerbach Publications, 2006.

[18] F. V. Barajas, A formal model for the building of state machines: A lightweight approach, *Proc. of Software Engineering Workshop*, pp.194-203, 2007.

[19] J. Jelena and G. Dragan, Achieving knowledge interoperability: An XML/XSLT approach, *Expert Systems with Applications*, vol.29, no.3, pp.535-553, 2005.

[20] S. Sendall and W. Kozaczynski, Model transformation: The heart and soul of model-driven software development, *IEEE Software*, vol.20, no.5, pp.42-45, 2003.

[21] T. Cargill, *C++ Programming Style*, Addison Wesley Professional, 1992.