Run-time Code Generators for Model-level Debugging in Domain-specific Modeling

DSM'16

Introduction

- DVRTS
  - Auto-adaptive run-time system
    - aimed at execution of control logic in the automation and robotics fields
  - Control logic is defined using function block language
    - Strucured Text (ST)
    - IEC61131-3
    - compiler for various hardware and software platforms
      - Intel
      - Arm
      - Windows, WinCE, Linux, Raspbian
  - Interpreter mode
    - virtual machine

Introduction

- DVRTS
  - Monitoring the execution of native code
  - Various metadata
    - variables datatypes
    - operation statuses
  - Various strategies for detection, documentation, and recovery from unexpected states
    - core element for model-level debugging
  - Update of control logic in the run-time
    - On-hot
  - Several communication channels
    - TPC/IP, named pipes, etc

Introduction

- Model-level debugging
  - Debugging program code on the platform-level is tedious and error-prone
    - obsolete techniques are used such as "printf" statements, data monitors
  - Applying DSM principles
    - raising debugging on the model-level
  - Further evolution of the DSM tools
    - model execution
  - Action reports language
    - MERL-like language
    - commands
      - feedback from the RTS
      - dynamic creating and updating representations of DSL concepts

RTCG

- Run-time code generators
  - Model-To-Text (M2T) transformations
    - incremental transformation of models
    - feedback from RTS

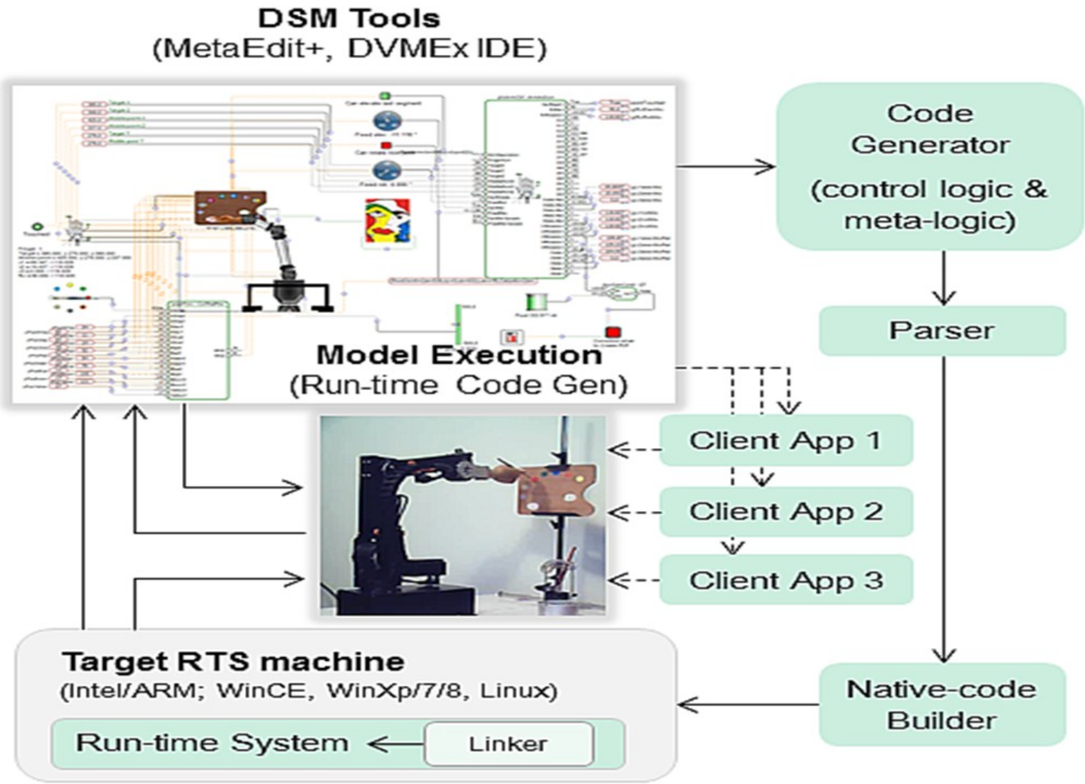


Figure 1. Role of RTCG-s in the DSM architecture.

RTCG

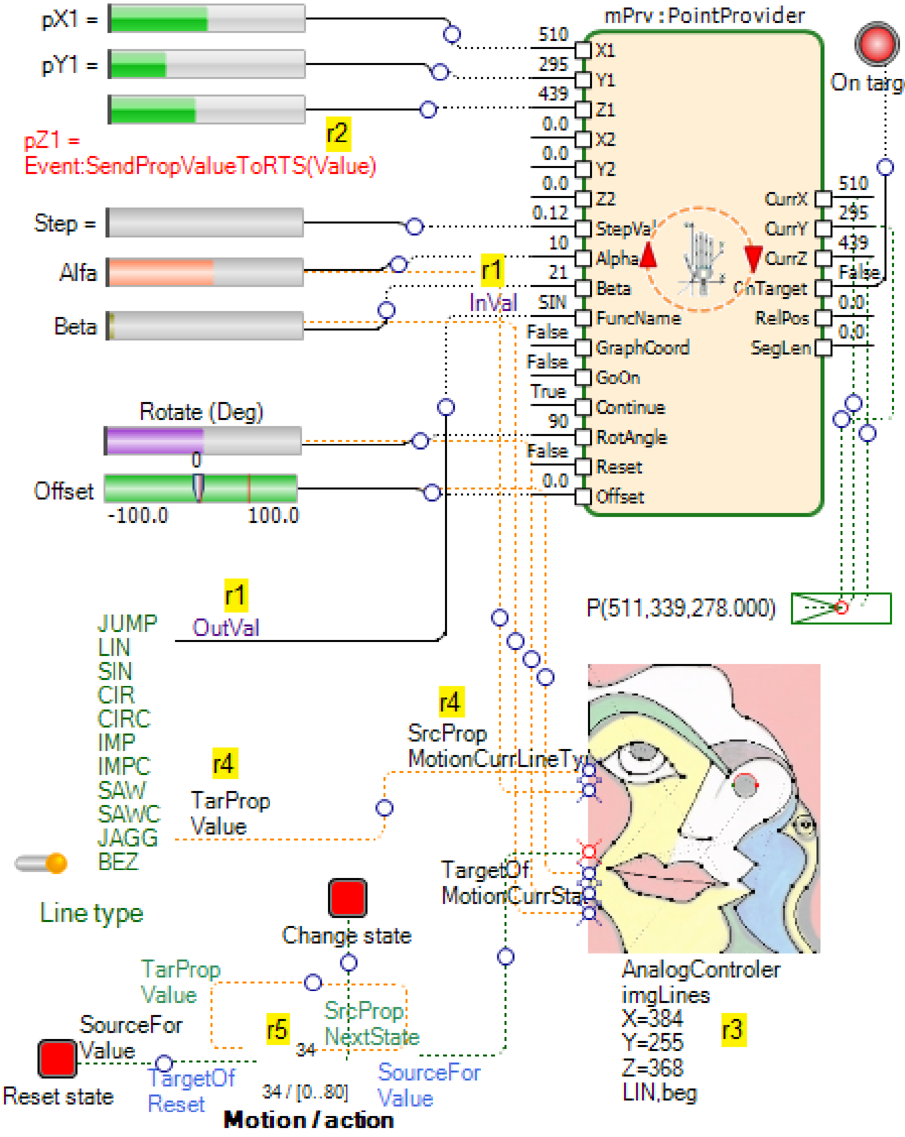
- Extensions
  - run-time construction of submodels
    - writing reports for the submodel construction
    - end-user views on a model
  - integration of various command languages and protocols
    - communication with the RTS
  - transactions
    - shifting the target RTS from one valid state to another
  - multi-client debugging
    - generation of end-user applications

Model-level debugging

- Three DSL
  - specification of control logic
    - similar to the function block language
  - specification of topological and mechanical features
    - properties of a robot arm
  - specification of an environment where control logic is executed

Model-level debugging

- Roles
  - *OutVal* and *InVal*
    - translated into CL code
      - *pPriv.X1 := pX1;*
    - evaluated on the RTS side
  - *SrcProp* and *TarProp*
    - exchanging properties
    - *swAction.Reset=stReset.Value;*
    - evaluated locally



Model-level debugging

- Properties
  - values
    - functions, expressions, events or reports
    - evaluated locally or on the RTS side
  - syntax
    - :.mPrav#FuncName.Value=$mList[$cnt];
    - :ConnPointAbsFor(,x);
    - :Left=ConnPointAbsFor(,x);
    - :SendPropValueToRTS(prName),1;

Model-level debugging

- Visual representation
  - important to have proper and functional representation as possible
  - arbitrary user component (control)
    - easily integrated within a modeling tool
    - meta-model extension
  - Properties
    - three groups
      - default representation in a modeling tool
        - » cannot be mapped to a property of some user component
      - directly mapped to one or more properties of one or more user components
      - belonging to user components
        - » not a part of the language definition

Model-level debugging

- Meta-model definition
  - XML form

```
<Type name ="TwoStateControler" id="DVLangObject">
<ctrlList>
<ctrl type="DVMExTwoStateSwitch" id ="ID" connProp
="Connections" dll="DVControl.dll" ns="DVMExControls">
<pList>
<prop name="ID" propType="Text" impName ="Name"/>
<prop name="PortAddress" propType="String" impName =""
label="HwPort" domain="String" defaultValue=""/>
</pList>
 </ctrlList>
```

Model-level debugging

- Synchronization of a model and the RTS
  - reporting language is extended
    - several commands
      - to synchronize a model and program code executed on the target system
  - RTS
    - several communication channels
    - command language is a main interface
    - synchronous and asynchronous
      - priority of command execution may be changed
    - various communication protocols
      - TCP/IP
      - message queues
      - named pipes

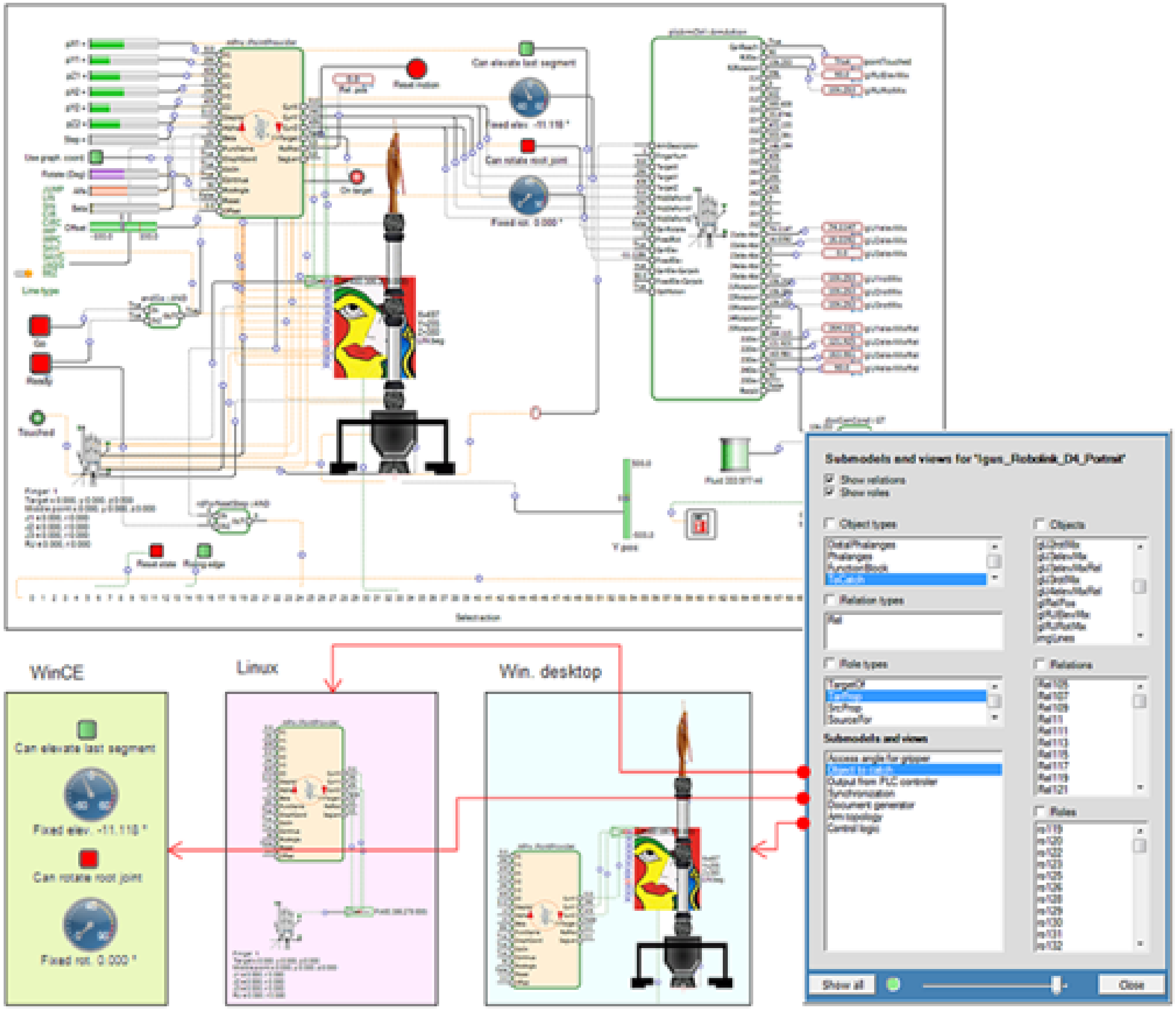
Model-level debugging

- Synchronization commands
  - begin_trans end_trans
    - enveloping set of commands
      - single transaction
  - webservice
  - f:external
    - executing an arbitrary command over the target RTS
      - using command language
  - function
    - invoking some built-in function
  - toset and tosetunique
    - transforming results of the command execution into a collection

Model-level debugging

- Multi-client debugging
  - user-driven activity
    - executing different dynamically created submodels
  - generating applications
    - using models and meta-models
  - each submodel becomes a separate application
  - client application sends commands to the RTS
    - using user components that the DSL concepts are mapped to
    - directly from hardware signals

Model-level debugging



Conclusion

- User components (controls)
  - easily integrate into modeling and metamodeling tools
    - already exist for various application domains
  - mapping of abstract and domain-specific model elements to platform-specific controls and properties
- Direct connection between modeling tool and a target system executing models
    - decrease need for writing separate program code for the debugging and simulation

Run-time Code Generators for Model-level Debugging in Domain-specific Modeling

DSM'16