SE SOFTWARE ENGINEERING

RWTH AACHEN UNIVERSITY

Management of Guided and Unguided Code Generator Customizations by Using a Symbol Table

27th October 2015

Pedram Mir Seyed Nazari, Alexander Roth, and Bernhard Rumpe

Software Engineering
RWTH Aachen

http://www.se-rwth.de/

Motivation

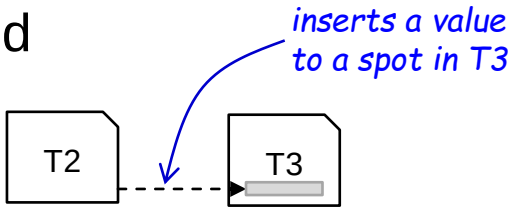
- In Model-Driven Development detailed code is generated from abstract models
  - models may be too abstract to describe every detail
- A possible solution:
  - customizations and adaptations of the code generator
- Basic customization of template-based code generation
  - directly adapt templates
  - disadvantage: affect all generated artifacts using this template

Contribution

- Two approaches for customizing templated-based generators
  - Guided approaches: restricted customization
  - Unguided approaches: unrestricted customizations
- Derive the information that should be managed
- Derive necessary extensions for template languages

Guided Customization Approaches

- Explicit declaration of spots that can be extended
  - e.g. variability points



- All other ways of customization are explicitly forbidden

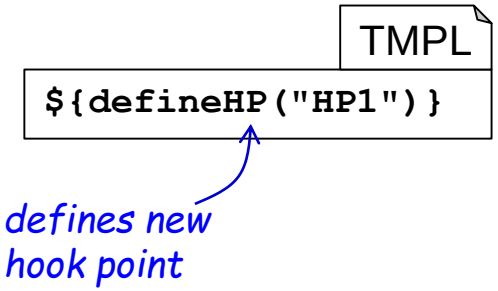
- Required main elements for template-based code generation
  - hook points: are uniquely identifiable spots defined for customization
  - hook points are set during the design time
  - content for each hook point needs to be bound explicitly
  - bounded values are either strings or other templates

- Advantage:
  - extension points are explicit and can be checked statically
- Disadvantage: might be too restrictive

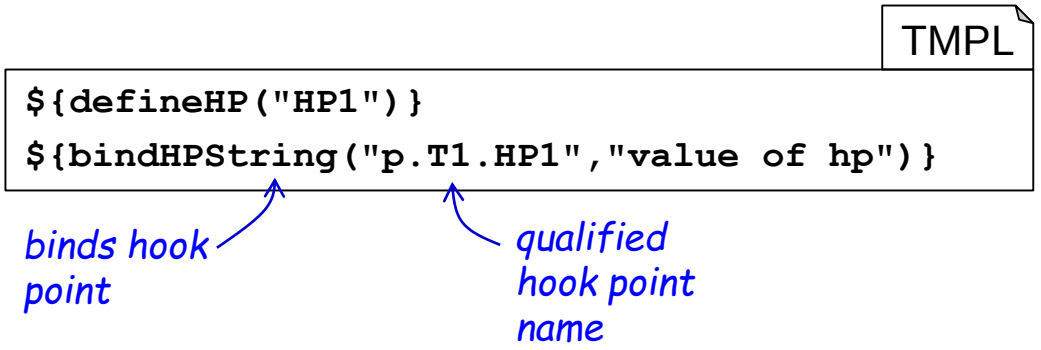
Requirements for Guided Customizations

- Extended template engine in order to define and bind hook points

Template T1



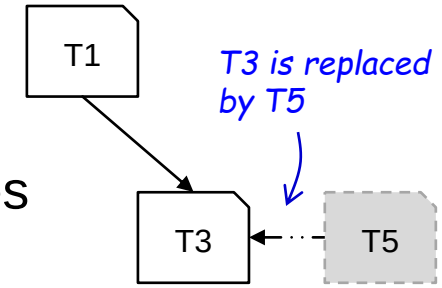
Template T2



- Requirements:
  - hook point is defined within the template and should be accessible from outside
  - given a (qualified name), the corresponding hook point definition must be obtained

Unguided Customization Approaches

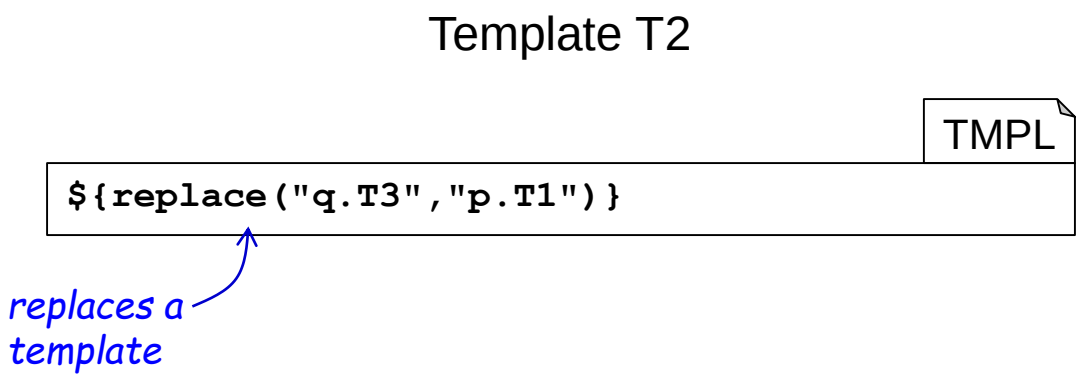
- Basic concept: directly customize the code generator by editing templates
  - replace templates: existing template is replaced
  - add before template: a template is added before an existing template
  - add after template: a template is added after an existing template



- Advantage: less restrictive than guided approaches
- Disadvantages:
  - tend to be more error prone
  - generator sources may not be available at generation-time
  - side effects as templates may be used in multiple places

Requirements for Unguided Customizations

- Extended template engine in order to replace templates
- Analogously for adding templates before or after existing templates



- Requirements:
  - replacing syntactically takes place in template T2. However, each template that includes T1 must be aware of this replacement
  - given a (qualified name), the corresponding template definition must be obtained

Requirements for Managing Customizations

- Resulting requirements for managing customizations:
  - manage information that is defined within that template and make it accessible (from outside)
  - manage information that is defined outside that template and make it accessible
  - given a reference, the corresponding definition must be obtained in order to access its associated information

- Goal:
  - Reuse existing infrastructures for managing customizations
  - Respect referential integrity

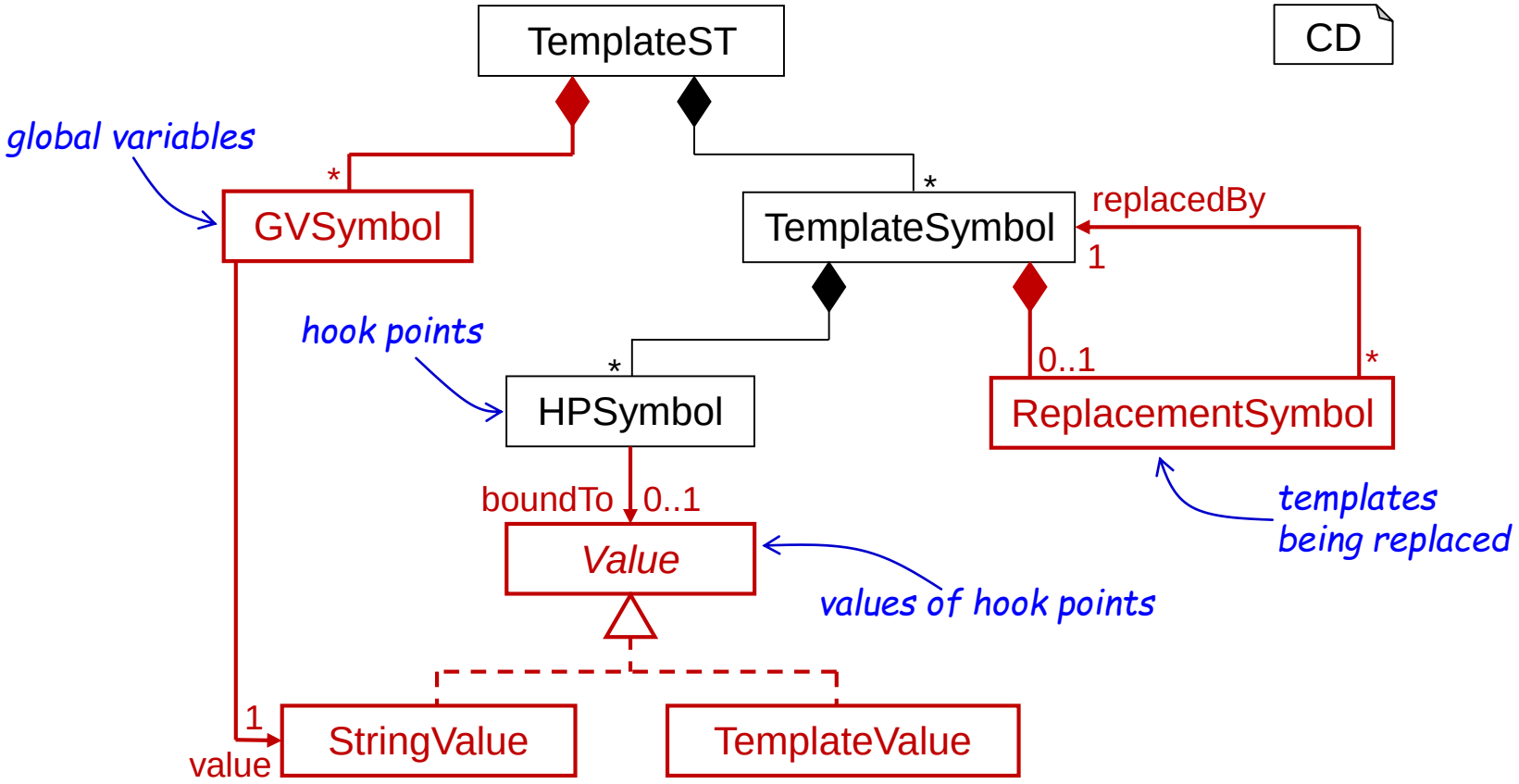
Symbol Table for Templates

- Symbol table naturally fits the requirements to manage code generator customizations

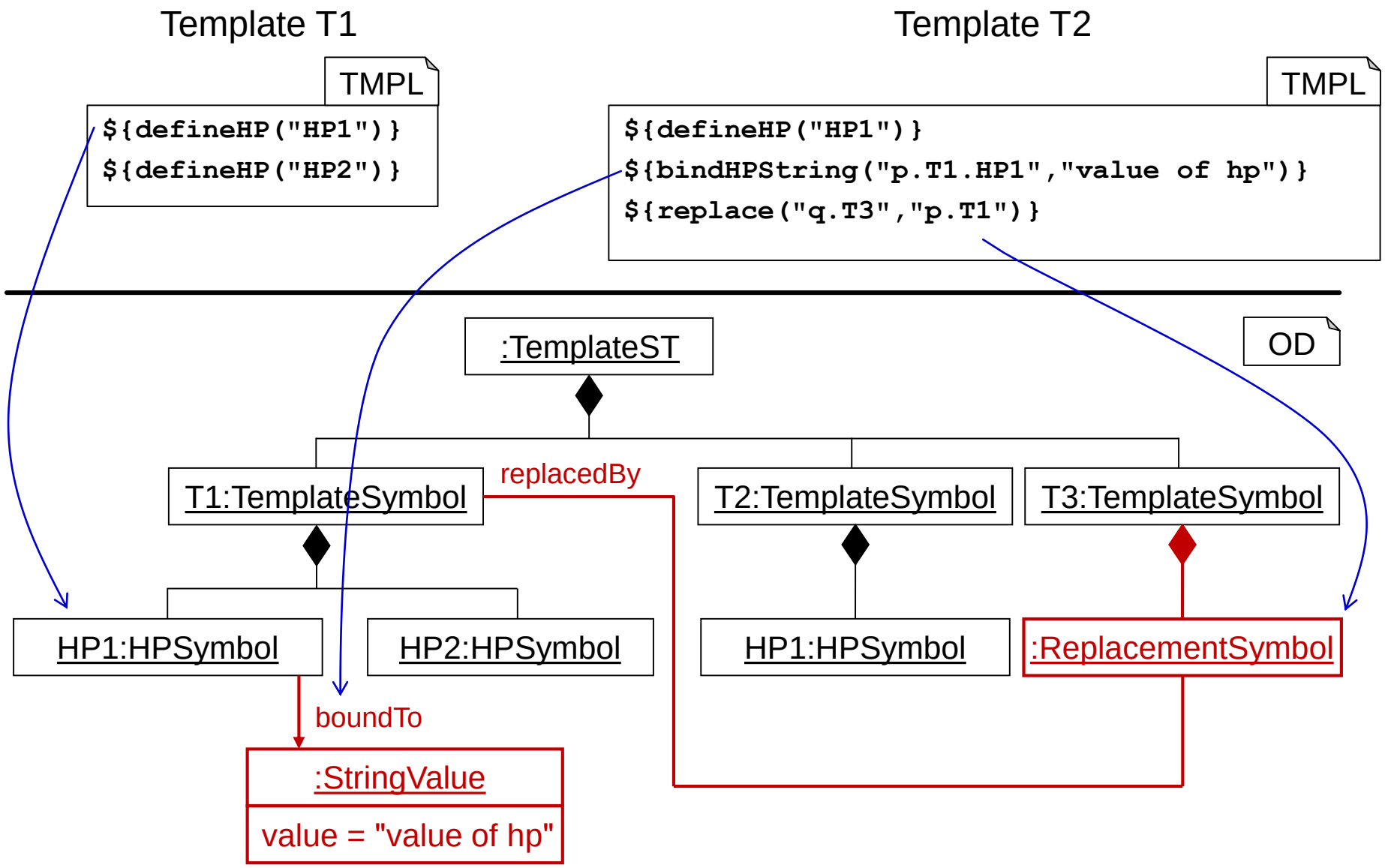
- Definition: Symbol table
  - it is a data structure that maps names to essential model elements
  - in MontiCore it may also represent the semantic meta model

- Symbol table can be extended to manage
  - hook point management
  - template customizations

Extended Symbol Table for Templates



Example


Template T1
TMPL
${defineHP("HP1")}
${defineHP("HP2")}
Template T2
TMPL
${defineHP("HP1")}
${bindHPString("p.T1.HP1","value of hp")}
${replace("q.T3","p.T1")}
:TemplateST
OD
T1:TemplateSymbol
replacedBy
T2:TemplateSymbol
T3:TemplateSymbol
HP1:HPSymbol
HP2:HPSymbol
HP1:HPSymbol
:ReplacementSymbol
boundTo
:StringValue
value = "value of hp"

Static and Dynamic Information

- Symbol table contains two types of information
  - static information: can be obtained without execution a template
  - dynamic information: can only be determined at generation-time
- By only analyzing templates:
  - template references and hook point symbols can be build
  - referential integrity can be checked
- Values bound to hook points can only be added at generation-time
  - values are not static and can change during execution
- A combination of static and dynamic information allows for efficient management of customizations

Conclusion

- Overview of guided and unguided customization approaches for template-based code generation
  - Goal: retrieve basic elements that need to be managed
- Provide requirements for a data structure to efficiently manage customizations
- Adapt the symbol table to manage customizations
  - Add hook points and template references
- Combine static and dynamic information for efficient management