Automatic Layout and Structure-Based Editing of UML Diagrams

Miro Spöнемann
Hauke Fuhrmann  Michael Matzen  Reinhard von Hanxleden

Real-Time and Embedded Systems Group, Department of Computer Science
Christian-Albrechts-Universität zu Kiel, Germany
{msp,haf,mim,rvh}@informatik.uni-kiel.de

M-BED 2010, March 12th
Outline

1. Pragmatics of MBE

2. Automatic Layout
   - Eclipse Integration
   - Algorithms

3. Structure-Based Editing
   - The Approach
   - Object Class Transformations

4. Conclusion
Pragmatics of MBE

- **Semantics** define the structure and meaning of a model
  - how to *interpret* it
Pragmatics of MBE

- **Semantics** define the structure and meaning of a model
  - how to *interpret* it
- **Syntax** defines the notation of a model
  - how to *visualize* it
Pragmatics of MBE

- **Semantics** define the structure and meaning of a model
  - how to *interpret* it
- **Syntax** defines the notation of a model
  - how to *visualize* it
- **Pragmatics** defines the interaction with a model
  - how to *edit* it
Pragmatics of MBE

- Textual editing is well known
Pragmatics of MBE

- Textual editing is well known
- Advanced editors offer versatile assistance for editing and formatting of text
  - 1-dimensional representation: relatively simple problem
Pragmatics of MBE

- Textual editing is well known
- Advanced editors offer versatile assistance for editing and formatting of text
  - 1-dimensional representation: relatively simple problem
- Graphical editing often limited to *drag-and-drop*
Pragmatics of MBE

- Textual editing is well known
- Advanced editors offer versatile assistance for editing and formatting of text
  - 1-dimensional representation: relatively simple problem
- Graphical editing often limited to *drag-and-drop*
- Little automated editing and formatting
  - 2-dimensional representation: complex problem
Resulting Problems

- A lot of time spent on the notation of models
  - Position new elements, arrange connections and surrounding elements...
Resulting Problems

- A lot of time spent on the notation of models
  - Position new elements, arrange connections and surrounding elements...
- High cost of model maintenance
Resulting Problems

- A lot of time spent on the **notation** of models
  - Position new elements, arrange connections and surrounding elements...
- High cost of model maintenance
- Solution: focus on the **structure** of models
Resulting Problems

- A lot of time spent on the notation of models
  - Position new elements, arrange connections and surrounding elements. . . 
- High cost of model maintenance
- Solution: focus on the structure of models

Perform structural modification, then let the computer do the layout!
Outline

1. Pragmatics of MBE
2. Automatic Layout
   • Eclipse Integration
   • Algorithms
3. Structure-Based Editing
   • The Approach
   • Object Class Transformations
4. Conclusion
Eclipse Integration

Edit Parts
Eclipse Integration

Miro Spönemann (University of Kiel)
Eclipse Integration

Edit Parts
File System
Domain Model
Notation Model
KGraph

Miro Spönemann (University of Kiel)
Layout and Editing for UML
Eclipse Integration

Miro Spönemann (University of Kiel)
Eclipse Integration

Edit Parts
File System
Domain Model
Notation Model
KGraph
Style
Annotation
KOption
KLayoutData
  width: 20
  height: 20

Miro Spönemann (University of Kiel)
Layout and Editing for UML
M-BED 2010 7 / 18
Eclipse Integration

Edit Parts
File System
Domain Model
Notation Model
KGraph
Style
Annotation
KOption KLayoutData
  width: 20
  height: 20
KOption
KOption
Layout
Algorithm

Miro Spönenmann (University of Kiel)
Layout and Editing for UML
M-BED 2010 7 / 18
Eclipse Integration

Miro Spönemann  (University of Kiel)
Eclipse Integration

Edit Parts
File System
Domain Model
Notation Model
KGraph
Style
Annotation
KOption
KLayoutData
  width: 20
  height: 20
  x: 324
  y: 117
KOption
KOption
Layout
Algorithm

File System

Notation Model

Domain Model

Miro Spönemann (University of Kiel)
Layout and Editing for UML
M-BED 2010
Extension Points

- Use Eclipse *extension points* to define an XML-based interface
Extension Points

- Use Eclipse *extension points* to define an XML-based interface
- Plug in new layout algorithms
Extension Points

- Use Eclipse extension points to define an XML-based interface
- Plug in new layout algorithms
- Configure default layout options
Extension Points

- Use Eclipse *extension points* to define an XML-based interface
- Plug in new layout algorithms
- Configure default layout options
- Register structure-based operations (presented later)
Extension Points

- Use Eclipse *extension points* to define an XML-based interface
- Plug in new layout algorithms
- Configure default layout options
- Register structure-based operations (presented later)
- The goal: offer the most suitable layout for
  - each type of diagram
  - each part of a structured diagram
Extension Points

- Use Eclipse *extension points* to define an XML-based interface
- Plug in new layout algorithms
- Configure default layout options
- Register structure-based operations (presented later)
- The goal: offer the most suitable layout for
  - each type of diagram
  - each part of a structured diagram

Layout works without any adaptions for most editors of the Eclipse Graphical Modeling Framework (GMF).
Connect existing implementations to our layout interface:
- Graphviz, a widely used command line tool
- Zest, part of the Eclipse Graphical Editing Framework (GEF)
- OGDF, a C++ library developed at the TU Dortmund
Graph Drawing Algorithms

- Connect existing implementations to our layout interface:
  - Graphviz, a widely used command line tool
  - Zest, part of the Eclipse Graphical Editing Framework (GEF)
  - OGDF, a C++ library developed at the TU Dortmund
- Specialized algorithms for specific diagram types
  - Developed a layouter for data flow diagrams, e.g. Simulink, SCADE, Ptolemy
  - OGDF has a special layouter for class diagrams
Data Flow Diagram Layout

Torq

>=

1.0

150.0

switch

0.0

Brake

X

KBRAKE

0.3

X

2.5

X

FBY

0.1

-0.1

<

VehicleSpeed

sub

TCYCLE

X

VehicleDynamic

MASSE

/

+
Outline

1. Pragmatics of MBE

2. Automatic Layout
   - Eclipse Integration
   - Algorithms

3. Structure-Based Editing
   - The Approach
   - Object Class Transformations

4. Conclusion
Structure-Based Editing

- Use a model transformation language to specify operations
  - E.g. Xtend (Eclipse M2T project)
Structure-Based Editing

- Use a model transformation language to specify operations
  - E.g. Xtend (Eclipse M2T project)
- Operate directly on the semantic model instead of the notation model
Structure-Based Editing

- Use a model transformation language to specify operations
  - E.g. Xtend (Eclipse M2T project)
- Operate directly on the semantic model instead of the notation model
- Perform automatic layout after each operation
Create a successor action in an activity diagram

```
Void createSuccessor(Action action):
    let newAction = new OpaqueAction:
    newAction.setActivity(action.activity) ->
    controlFlow(action, newAction);

Void controlFlow(Action action1, Action action2):
    let flow = new ControlFlow:
    flow.setActivity(action1.activity) ->
    flow.setSource(action1) ->
    flow.setTarget(action2);
```
Object Class Transformations

- The UML metamodel has many specializations
  - E.g. OpaqueAction, CallBehaviorAction, CallOperationAction, CreateObjectAction, AcceptEventAction, SendSignalAction...
Object Class Transformations

- The UML metamodel has many specializations
  - E.g. OpaqueAction, CallBehaviorAction, CallOperationAction, CreateObjectAction, AcceptEventAction, SendSignalAction...
- To change an instance, remove it, add the new instance, and fix all properties and connections
Object Class Transformations

- The UML metamodel has many specializations
  - E.g. OpaqueAction, CallBehaviorAction, CallOperationAction, CreateObjectAction, AcceptEventAction, SendSignalAction...
- To change an instance, remove it, add the new instance, and fix all properties and connections
- Structure-based editing can be used to simplify this
Object Class Transformations

- The UML metamodel has many specializations
  - E.g. OpaqueAction, CallBehaviorAction, CallOperationAction, CreateObjectAction, AcceptEventAction, SendSignalAction...
- To change an instance, remove it, add the new instance, and fix all properties and connections
- Structure-based editing can be used to simplify this
- Write **toggling** operations that perform all these steps automatically
Conclusion

- Introduced a framework for automatic layout in Eclipse
Conclusion

- Introduced a framework for automatic layout in Eclipse
- Perform structural operations employing automatic layout
  - Structure-based editing

Concepts can be applied with minimal effort to all GMF diagram editors, e.g., SyncCharts, a synchronous Statecharts dialect available in the open source project KIELER.
Conclusion

- Introduced a framework for automatic layout in Eclipse
- Perform structural operations employing automatic layout
  - Structure-based editing
- Concepts can be applied with minimal effort to all GMF diagram editors
  - E.g. SyncCharts, a synchronous Statecharts dialect
Conclusion

- Introduced a framework for automatic layout in Eclipse
- Perform structural operations employing automatic layout
  - Structure-based editing
- Concepts can be applied with minimal effort to all GMF diagram editors
  - E.g. SyncCharts, a synchronous Statecharts dialect
- Available in the open source project KIELER
Future Work

- Define a concrete set of transformations on the UML metamodel

Evaluate resulting operations in the context of actual development projects

Synchronize graphical diagrams with textual representations

View management: dynamic creation of graphical views

Display models with different levels of detail
Future Work

- Define a concrete set of transformations on the UML metamodel
- Evaluate resulting operations in the context of actual development projects
Future Work

- Define a concrete set of transformations on the UML metamodel
- Evaluate resulting operations in the context of actual development projects
- Synchronize graphical diagrams with **textual** representations
Future Work

- Define a concrete set of transformations on the UML metamodel
- Evaluate resulting operations in the context of actual development projects
- Synchronize graphical diagrams with textual representations
- View management: dynamic creation of graphical views
  - Display models with different levels of detail
View Management: UML Metamodel
View Management: UML Metamodel
Contact

www.informatik.uni-kiel.de/rtsys