Model Driven Integration of Software Systems

Alek Radjenovic

HISE Seminar
18 Oct 2011

Outline
- Context & Problems
- Key objectives
- Solution
  - Concepts
  - Implementation
- Case studies
Why model?

- We’ve always modelled
- simplify the reality
  - better understand the world around us
- design and develop objects/systems in a
  gradual/incremental fashion using structured approach
- apportion various aspects of design to the ‘experts’ (a.k.a. domain specialists)
  - used both in development as well as analysis
- Humans use abstraction to better deal with complexity
  - by excluding the unnecessary detail
- Abstraction (of a system, or its environment, or both)
  - is the key tool in modelling

Software modelling: a historical perspective

- 50’s and 60’s – abstraction of 0’s and 1’s into:
  - symbolic names, high-level operations, block structures, ...
- 70’s – system design
  - functional isolation, API, graphical representation, ...
- 80’s – OO, software architecture
- 90’s – MDE/MBDE
  - UML, ADLs, DSLs, ...
Software modelling: a modern view

- (almost) all software artefacts are models
  - requirements specifications
  - design (software architecture) specifications
    - e.g. UML diagrams
  - source code
  - database schemas
  - petri nets
  - even a directory listing from your hard disk!

Context & Problems

- Large scale, complex, software → Distributed teams → Heterogeneous system components (legacy and new) → Late, sometimes unpredictable, costly system integration
- Increased use of model driven engineering + Multiple (standard and non-standard) modelling platforms + Varying versions of modelling tools and languages = Inability to integrate easily at model level
- Integration at modelling stage → Early detection of incompatibilities → (potentially huge) cost saving during system integration
Ultimate objective: Model Integration

- Questions to answer:
  - Is model integration possible?
  - To what extent?
  - How to deal with identified incompatibilities?
- Beyond scope
  - Model maintenance; Model evolution
- Blue-skies research for SSEI
  - not ‘without a clear goal’, but certainly ‘curiosity-driven science’

State-of-the-art

- Model management techniques
  - Model compatibility
    - initiative for a unifying model (e.g. MOF)
    - theoretically – a step in the right direction
    - in practice – lack of widespread support by tools; versions
  - Model comparison and differencing
    - first 2 steps in assessing incompatibilities
    - SiDiff – graphical representations (UML, Simulink)
    - EMF Compare – Eclipse (EMF input models)
State-of-the-art (2)

- Model Transformation
  - ATL – Eclipse based; robust tool support
    - language somewhat cumbersome
    - substantially declarative nature (e.g. iterations over complex structures not easy)
  - ETL – Eclipse based (Epsilon)
    - hybrid (declarative & imperative)
    - imperative part: loops, assignments, statement sequencing
    - reuses a portion of OCL for navigating model elements
  - VIATRA2 – UML; graph transformations & ASMs

State-of-the-art (3)

- Model Composition (Merging)
  - AMW (ATLAS Model Weaver) – Eclipse based
  - EML (an Eclipse/Epsilon language)
  - ReuseWare – for Semantic Web (OWL, Xcerpt, XQuery)
  - openArchitectureWare – Eclipse based
  - Kermeta – meta-programming environment based on MOF (Eclipse based tool)

- Conclusion
  - no integrated approaches
  - key issue: different meta-models (languages)
Conceptual approach

- Decomposition of models
  - Structural (mostly evident at syntax level)
  - Behavioural (semantics)
- Compatibility checking
  - Structural: static; production of correspondence models
  - Behavioural: dynamic; simulation + (post-mortem) analyses

Solution in a nutshell

- Input models
  - Heterogeneous (theoretically any modelling platform)
  - Textual representation
- Interchange platform
- Compatibility checking
  - Structural
  - Behavioural
- Model integration
- Proof of the pudding: large case study (avionics)
Solution platform

- SMILE (Simple Model Integration Languages with Execution engines)
- Three key components (languages + tools):
  - SMILE-S (model structure)
  - SMILE-X (model behaviour)
  - SMILE-I (model integration)
- Key objectives
  - ‘meta-model agnostic’
  - extensible
  - scalable

SMILE-S as a black box

- Philosophy: minimal core, extensible
- External to the core (simple evolution):
  - Pattern language
  - Rule language
  - Language parsers
    - one way transformation
  - Libraries of routines
    - for use by patterns and rules
  - 2 input models only
SMILE-S: Internal architecture

- INPUT MODEL 1
- PARSER 1
- SMILE TREE 1
- PATTERNS
- RULES
- EXTERNAL LIBRARIES
- COMPATIBILITY CHECKING
- CORRESPONDENCE MODELS
- INPUT MODEL 2
- PARSER 2
- SMILE TREE 2

SMILE-S: Interchange format

- tree-based
  - nodes = model elements, links = containment relationship
  - attributes: name, native type, native ID, SMILE type, SMILE ID
  - properties: dependent on the model

<table>
<thead>
<tr>
<th>node</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTES [KEY] = VALUE</td>
</tr>
<tr>
<td>PROPERTIES (optional)</td>
</tr>
<tr>
<td>PROD = (VALUE, TYPE)</td>
</tr>
<tr>
<td>CHILDREN (optional)</td>
</tr>
<tr>
<td>C1, C2, ... : node</td>
</tr>
</tbody>
</table>
SMILE-S trees

SMILE-S: Plug-ins/Extensions
- (language/model) Parsers
- Meta-model knowledge displaced externally
- Comparison/Matching/Compatibility routines
  - typically return Boolean values
SMILE-S: Patterns

- Specify constellations of
  - 1..* nodes
  - 0..* ‘filters’ (match criteria)
    - Y.Property[xmi:type] = uml:class
  - Applied to a single model

SMILE-S: Rules

- Specification
  - left and right pattern correspond to input models
  - (a set of) matching conditions
    - Boolean logic + internal/external routines
- Application
  - returns a pair of compatible nodes/subtrees (or NULL)
  - has attached
    - a similarity score
    - trace
SMILE-S: Correspondence models

- Colour coded trees
- Provide visual guidance to incompatibilities
- Contextual information details failed rules

SMILE-S: Summary

- Interchange format (trees)
- Pattern language
- Rule language
- Execution engine
- Correspondence models
- Plug-ins
  - parsers:
    - XMI (UML, SysML), Java, (modified) SQL, a DSL (DOS directory listing)
  - comparison libraries
SMILE-X as a black box

SMILE-X: Internal architecture

- Behavioural model combines template artefacts with model elements
- effectively, a meta-model which needs to be instantiated before we can run the simulation
- dynamic creation and deletion of objects not a feature (yet)
SMILE-X: Behaviour

- Basic unit of behaviour
- Messages (input or output)
- Operations
  - have duration (e.g. WCET)
  - collection of actions
  - can have condition (just a Boolean label - evaluates to both true and false during simulation)
  - can be linked with action

SMILE-X: Simulation objects

- SMILE-S tree elements + Behavioural template = Behavioural Model
  - a collection of behavioural elements
- Behavioural model + Configuration = Simulation Model
  - a collection of simulation objects
SMILE-X: Triggers

- A (compound) Boolean expression
- Flag to stop execution or continue
- Example:

```
<table>
<thead>
<tr>
<th>Object</th>
<th>Operator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller State</td>
<td>==</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>Controller Input</td>
<td>==</td>
<td>LIFT_REQUEST</td>
</tr>
<tr>
<td>Controller Output</td>
<td>==</td>
<td>MOVE</td>
</tr>
</tbody>
</table>
```

SMILE-X: Schedules

- Simple activation

```
EXECUTE ACTIONS
INCREMENT TIME
```

- Double buffer

```
EXECUTE ACTIONS INTO BUFFERS
CONFLICT RESOLUTION
INCREMENT TIME
```

- Event based

```
COMPUTE FUTURE EVENTS
ACTIVATE NEXT EVENTS
```
SMILE-X: Traces

- Sequential
- Provide information on:
  - input message
  - failed conditions
  - executed actions
  - output message
  - triggered conditions
- Colour coded

SMILE-X: Summary

- Uses SMILE-S models (trees)
- Adds semantics to nodes and properties
- Uses a template to create behavioural model:
  - maps between behavioural concepts and structural elements
  - specifies allowed interactions
- Manual instantiation of behavioural model elements to create simulation objects
- Trigger specification (manual)
- Schedule selection
- Trace inspection
SMILE-I: Integration component

- Small, in-house case studies
  - File system (DOSDir DSL) + Documents database (SQL)
  - Lift system (UML interaction diagrams)
- Model integration
  - based on model weaving
  - examples very basic
    - need real-world case studies

SMILE-I: Example

- Before: 4 physical files, 3 database records (rows)
- After: 4 files, 4 rows
SMILE-I: Model weaving

- Requires additional plug-in for each language
- Key operations (parameters):
  - creation, deletion (node/subtree)
    - based on unmatched patterns
  - insertion (insertion point/parent node)
  - move (node/subtree, new parent)
    - manual intervention (at the moment)
    - future: rule-based

Tools: Implementation summary

- GUI driven
- Microsoft Windows
- C# and .NET
- Extensible via plug-ins
- Project based
Case study: Airport (legacy)

Case study: Airport (new)
Case study: behaviours

Air Analyser

Case study: behaviours

Police Unit

Case study: behaviours

Fire Department
Case study: behaviours
ACM (Airport Crisis Management unit)

Refereed papers
Reports

- [SSEI-TR-0000015] "State-of-the-art Survey of Model Driven Integration of Software Systems"
- [SSEI-TR-0000042] "Structural Compatibility in Models"
- [SSEI-TR-0000043] "Behavioural Compatibility in Models"
- [SSEI-TR-0000095] "Model Driven Integration: CMS-1 CS Case Study"

Links

- SiDiff - [http://sidiff.org/](http://sidiff.org/)
- ReuseWare - [http://www.reuseware.org/](http://www.reuseware.org/)
- openArchitectureWare - [http://www.openarchitectureware.org/](http://www.openarchitectureware.org/)
- Kermeta - [http://www.kermeta.org/](http://www.kermeta.org/)