Integrating Specifications: Development Relations and Correspondences

Eerke Boiten
Computer Science Department
University of Kent at Canterbury

INT @ ETAPS 2002
Research area & approach

- Viewpoint Specification (*partial specification*)
- Formal Methods (Z, CSP, →UML)
- Open Distributed Processing framework
- Consistency & Unification

- Cross Viewpoint Consistency in ODP
- ODP Viewpoints in a Development Framework
- A Constructive Framework for Partial Specification
- with John Derrick, Marius Bujorianu and others
Integration vs. Consistency

- **Integration** of two descriptions: a single description that faithfully represents both
- **Consistency** of two descriptions: posing no contradictory requirements
- **Unification** of two descriptions: combination of their requirements
  = constructive consistency = integration

- **Integration** of description *formalisms* = ...
### Dimensions of consistency

<table>
<thead>
<tr>
<th>Homogeneous</th>
<th>Structural</th>
<th>Behavioural</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Solved</td>
<td>Solvable</td>
</tr>
<tr>
<td></td>
<td>(modules static semantics)</td>
<td>per language</td>
</tr>
<tr>
<td>Heterogeneous</td>
<td>Solved</td>
<td>Difficult!</td>
</tr>
<tr>
<td></td>
<td>(case tools)</td>
<td></td>
</tr>
</tbody>
</table>
Homogeneous Behavioural Integration

- Model/implementation based: integration = common implementation.
- Disadvantages: not abstract, not incremental, not relevant (feedback!)
- Alternative: use development relations
Homogeneous Development Relations

- “refinement” relations, often predefined
- normally pre-orders: transitive & reflexive
- possibly from models/implementations: \( x \) develops \( y \) iff \( \text{Models}(x) \subseteq \text{Models}(y) \)
- unification = (“greatest”) lower bound
- consistency = \( \exists \) unification
Not Quite as Homogeneous …

• Multiple development relations for single notation, e.g. process algebra
• Pairs \((spec, \leq)\) as primitive partial specifications
• Unification, consistency still possible
• Can embed into single development relation

• Assumption: shared interface, unrealistic
Correspondences

- (ODP) Linking viewpoints
- Meta-level, structural: for every X in spec. 1, there should be a matching Y in spec. 2.
- Spec. level: which Y matches a particular X?

- Primitive: give X and Y same name
- Qualified naming
- Data dictionary
Interface or Internals?

- Some names in a specification belong to interface
- Other names do not belong to observable behaviour

- Correspondence relating interfaces: stable
- Correspondence relating internals: mutable
- Correspondence relations: names + … … …
Integration & Correspondence

- Dependent on shape of correspondences in the language
- Correspondence becomes part of integrated specification:
  - E.g. data dictionary $\exists$-quantified
  - Equivalences and other functional correspondences lead to substitutions
- Consistency depends on correspondence
On to Heterogeneity …

• Still pairs \((spec_i, dev_i)\)
• Definition of integration, consistency: same.
• Consequence: \(dev_i\) need common co-domain. (Common semantics/implementation.)

• Correspondence: bridges formalisms!
• Must relate interfaces. How?
Heterogeneous Correspondences

- Correspondences must operate on common level?

- If $dev_i = dv_i \; ; \; trans_i$ for homogeneous $dv_i$ (all $i$) then correspondences can be at specification level. (Internals related $\Leftrightarrow$ preserved by $trans_i$)

- Can also embed this into homogeneous model.
General Abstract Nonsense?

- Integration *only* possible if preservation of requirements ensured.
- Preservation of requirements must be verifiable: either directly (refinement) or indirectly (comparing semantics).
- Inclusion of requirements at semantic level is the *definition* of refinement. Unavoidable!
Applications

- Z, process algebra, and combinations
- UML?
- Structural consistency OK.
- Main issue: notion of implementation ~/conformance for all included notations.
- Development “for free”.

E.A.Boiten@ukc.ac.uk
(c) 2002, Computer Science Dept., The University of Kent at Canterbury
Practical Approaches

- Component approach: a component refined by every system which includes it and utilises it according to contract.

- Layered approach: templates. Templates refined by every consistent instantiation.

- Both lead to obvious integration and simple consistency check.

- But: aspects/viewpoints/non-functional requirements mantra: cheap not always possible.
Abstract Approach

- Defining a high-level common semantics for a large collection of languages. (Category theory, institutions, transformation systems?)
- Translations into semantics (injective?)
- Investigating language properties and features that make partial specification feasible, constructive, practical. (Pushouts, pullbacks, order preserving translations, …)
Conclusions

• No silver bullet

• Integration of formalisms: identify semantics, refinement, translation, correspondence, interfaces. (Many inter-relations.)

• Framework, vocabulary?