Reflections on the Anatomy of Software Composition Languages and Mechanisms

Michel Chaudron
Department of Mathematics and Computing Science
Technische Universiteit Eindhoven, The Netherlands
m.r.v.chaudron@tue.nl
Aims of Research

Development of techniques that enable the building and maintenance of large software systems through the composition and decomposition of existing pieces of software.

Subgoal: Understand software (de)composition mechanism.
Structure of this talk

1. Introduction
2. Model for Software Composition
3. Examples of Software Composition Mechanisms
   - of different scale / complexity
   - what are their strengths and weaknesses
     → recommendations for design
4. Proposal for Composition Mechanism
5. Concluding Remarks
Terms and Definitions

Data flow = structure/geometry and dynamics of flow of data between components

Control flow = structure/geometry and dynamics of flow of information related to the management of activities within and among components

Information model = agreements on representation (syntax) and meaning (semantics) of data

Component = executable software item that is (to be) composed
Model for Software Composition

virtual component interaction

integration of data flow
integration of control flow
integration of information model

composition mechanism

component interaction

control flow
data flow

component

component

data flow
control flow
Examples of Composition Mechanisms: Programming Languages

- **Component**
  - Assignment (statement/procedure)

- **Control Flow**
  - if, then, while, sequence, ...
    - Implicitly manipulate program counter

- **Data Flow**
  - Implicit through sharing of variables

- **Information Model**
  - Syntax: program syntax
  - Semantics: left to programmer
    - + type checking
Reflection on Programming Languages

- Composition mechanism is subservient to computational paradigm (2nd class citizen)

- Computation and composition are strongly intertwined. Difficult to change one without changing the other

- Composition mechanism is fixed
Examples of Composition Mechanisms: Pipe and Filters

E.g. Unix: `ls | grep ... | sort | print`

- Component
- Control Flow
- Data Flow
- Information Model
- Executable (program)
- ... leftopen
- (linear) stream
- Syntax: ASCII
- Semantics: Programmer

TU/e
Reflection on Operating Systems & Pipe and Filter

Pipe and Filter:

• Composition is exogenous
  No need to change components when composed differently

• Composition is non-intrusive
  Behaviour of a component does not depend on how it is composed
Examples of Composition Mechanisms: 

- Middleware are

- (distributed) programs

- Multiple interaction styles (see next)

- Representation: IDL

- Semantics: domain models

  - (finance, transport, telecom, medical, ... )
Example: Remote Procedure Call

virtually component interaction

component interaction

control flow

data flow

parameters

integration of data flow

integration of control flow

integration of information

component

component

control flow

data flow

control flow
Example: Remote Procedure Call

- Integration of data flow
- Integration of control flow
- Integration of information
- Virtual component interaction
- Component interaction
- Control flow
- Data flow
- Results
Reflection on RPC

Coupling of data-flow and control flow introduces unnecessary coupling.

Develop mechanisms for dealing with data-flow and control flow separately.

Capture idiom:

- Control composition:
  - exception handling
- Data composition:
  - caching
  - replication
Interaction Styles Induce Coupling

Time-coupling:
source and destination have to agree on acting in time (synchronization)

Spatial-coupling:
source and destination have to agree on location/identity
(‘name space’ is only an indirection)

<table>
<thead>
<tr>
<th>independent</th>
<th>dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>“undirected”</td>
<td>“directed”</td>
</tr>
<tr>
<td>“anonymous”</td>
<td>“synchronous”</td>
</tr>
</tbody>
</table>

- broadcast
- radio
- peer to peer
- telex
- publish/subscribe
- magazine
- store & forward
- e-mail

Independent vs. Dependent

“asynchronous” vs. “synchronous”
Reflection on Interaction Styles

Which Interaction Styles to Use?

Principal approach:
CBD is best served with minimization of coupling:
Technologically optimal: publish-subscribe
Aim for world domination??

Pragmatic approach:
accept heterogeneity of interaction styles
and develop integration/composition technology
Examples of Composition Mechanisms:

Internet's World Wide Web

WWW = blackboard; applications can get and put pages

WWW = pages/applications

control & data flow

information model

representation: XML, HTML
semantics: many, e.g. e-commerce,
Reflection on World Wide Web

- Allows easy addition/removal with low impact on overall system behaviour

- Is a composition mechanism of high algorithmic complexity

- Composition mechanisms can be subject to non-functional requirements
  - scalability, extensibility
  - performance, reliability
Proposal: Components without Style

Motivation:

• Avoid architectural mismatch
• Increase reuse of component
• Easy changing of styles (localise impact of change)
• Ease wrapping
• Connectors as first-class citizens
Proposal: Components without Style

- **Component**: Purely computational entity, interaction style not predetermined by computation, e.g., Gamma model (Banatre & Le Métayer)
- **Style**: Wraper that imposes a particular interaction style, e.g., Piccola
- **Messaging layer**: Basic mechanism for forwarding messages
Concluding Remarks

- Composition languages are the language interface to (de)composition mechanisms.

- Software composition occurs at different orders of scale. Composition mechanisms may have high algorithmic complexity.

- A software composition language should provide mechanisms for:
  - integration of control flow,
  - integration of data flow,
  - integration of information model

- Separating means for integration of control-flow and of data-flow:
  - avoids implicit coupling
  - increases expressivity
Thank you for your attention

Questions?
Discussion

- Composition is more than binding

- Composition mechanisms should allow decomposition without unacceptable disturbance to the system

- Composition mechanisms should be composable to form higher level abstractions

- Components and composition models should be designed in unison (practical or pragmatic?)

- Components should be robust against non-sensical input

- When composition strongly influences characteristics
Examples of Composition Mechanisms:

Operating System

- **Component**
  - executable (program)
- **Control Flow**
  - scheduler/synchronization
- **Data Flow**
  - messaging/files
- **Information Model**
  - syntax: ASCII
  - semantics: programmer

**OS:** Mechanism succeeds in addition/removal of 'components'