The ToolBus
a service-oriented architecture for language-processing tools

Paul Klint
Motivation

• The **ASF+SDF Meta-Environment** is an IDE for developing language processing tools:
  – implementation of domain-specific languages
  – software analysis & transformation (renovation)
  – technology used by ING, IBM, Fortis, ASML, Getronics, Lucent, and research institutes

• The **ToolBus** is a coordination architecture invented to improve internal structure and maintainability of the Meta-Environment
Motivation

• ToolBus is a SOA *avant la lettre*
• What can we learn from other SOAs?
• What are the lessons learned from using the ToolBus?
• What can be improved?
• What new options are enabled?
• A status report on current ToolBus-related developments
Roadmap

• Two examples of component-based systems
• Brief history of software components
• Coordination languages & SOA
• The ToolBus technology
• Next-generation ToolBus
  – regaining the costs of componentization
  – user-interface untangling
  – features under consideration
Roadmap

- Two examples of component-based systems
- Brief history of software components
- Coordination languages & SOA
- The ToolBus technology
- Next-generation ToolBus
  - regaining the costs of componentization
  - user-interface untangling
  - features under consideration
Example 1

12 Plug types A-L
Solutions
The result

Standardisation of electricity outlets failed

Adapters ensure compatibility
Example 2: the screw
History of the screw

- 400 BC invention of screw
- 18th century metal screws
- 1797: Henry Maudslay patents the Lathe

Archytas of Tarentum
(428 BC - 347 BC)
Standaardisation of screw thread

- 1841: England switches to thread by Joseph Whitworth
- 1864: William Sellers improves this design; recommended by US navy
- 1880: Sellers' thread in general use

Standaardisation takes ages (400 BC – 1880)

Software is only 70 years old!
Roadmap

- Two examples of component-based systems
- Brief history of software components
- Coordination languages & SOA
- The ToolBus technology
- Next-generation ToolBus
  - regaining the costs of componentization
  - user-interface untangling
  - features under consideration
Roadmap

• Two examples of component-based systems
• Brief history of software components
• Coordination languages & SOA
• The ToolBus technology
• Next-generation ToolBus
  – regaining the costs of componentization
  – user-interface untangling
  – features under consideration
Vision: Softwarecomponents

- “Mass produced software components” (1968)
- Describes an industry of software components
- Variability: precision, robustness, time/space considerations, ...

How has this vision materialized?

What is the current state of affairs?

Dough McIlroy
## Structuring and Composition of Software

<table>
<thead>
<tr>
<th>Statements</th>
<th>Procedures</th>
<th>Modules</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured programming</td>
<td>Functions, procedures &amp; libraries</td>
<td>Object-oriented programming &amp; Modules</td>
<td>Coordination languages &amp; SOA</td>
</tr>
<tr>
<td>Unix pipes</td>
<td>DCOM</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ken Thompson & Dennis Ritchie

- Unix 1972
- Pipes (contribution McIlroy)

Ken Thompson & Dennis Ritchie
Unix Pipes

Notation: P1 | P2

Example: who | wc -l

who: gives list of current users; one per line
wc (wordcount): counts number of lines in a file (-l)

Counts the number of users
Unix Pipes

- Unix pipes allow the connection of programs in a **linear pipeline**
- Common type of input and output: **string of characters**
- Programs have to **parse** their input when more data have to be exchanged
Roadmap

- Two examples of component-based systems
- Brief history of software components
- Coordination languages & SOA
- The ToolBus technology
- Next-generation ToolBus
  - regaining the costs of componentization
  - user-interface untangling
  - features under consideration
Roadmap

- Two examples of component-based systems
- Brief history of software components
- Coordination languages & SOA
- The ToolBus technology
- Next-generation ToolBus
  - regaining the costs of componentization
  - user-interface untangling
  - features under consideration
Coordination languages

“Coordination languages and their significance” (1992)

Connect programs via a common tuplespace

Nicolas Carriero and David Gelernter
Coordination languages

“Coordination languages and their significance” (1992)

Connect programs via a common tuplespace

Requires synchronisation!

Nicolas Carriero and David Gelernter
ToolBus coordination architecture

The ToolBus: a component interconnection architecture (1994)

Connect programs via active processes instead of a passive blackboard

Jan Bergstra and Paul Klint
Other approaches to coordination in the same period ...

Robert van Liere

Datamanager
common data on a blackboard

Farhad Arbab

Manifold
parallel processes with fixed communication channels
Observations (1)

Structured programming, procedures, modules, and OO do not help at the system level.

Procedures introduce spaghetti at the system level.
Observations (2)

• By connection different software components systems become heterogeneous:
  – different languages (C, Java, C#)
  – different platforms (Linux, J2EE, .NET)

• Due to local network (LANs) and multi-processors-on-a-chip systems become more and more parallel/distributed
Questions

- How do we ensure interoperability?
- How do we exchange data?
- How can components invoke each other?
- How can we integrate different user-interfaces?
Service-oriented Architecture

- Loose coupling
- Service contract
- Autonomy
- Abstraction
- Reusability
- Composability
- Statelessness
- Discoverability

- Message exchange patterns
- Coordination
- Atomic transactions
Roadmap

• Two examples of component-based systems
• Brief history of software components
• Coordination languages & SOA
• The ToolBus technology
• Next-generation ToolBus
  – regaining the costs of componentization
  – user-interface untangling
  – features under consideration
Roadmap

• Two examples of component-based systems
• Brief history of software components
• Coordination languages & SOA
• The ToolBus technology
• Next-generation ToolBus
  – regaining the costs of componentization
  – user-interface untangling
  – features under consideration
Coordination, Representation & Computation

- **Coordination**: the way in which program and system parts interact (proc. calls, RMI, ...)
- **Representation**: language & machine neutral data exchanged between components
- **Computation**: program code that carries out a specialized task

A rigorous separation of coordination from computation is the key to flexible and reusable systems
Architectural Layers

Coordination

Representation

Computation

Single Component

Representation

Computation

Single Component

Cooperating Components
Generic Representation
Annotated Terms (ATerms)

• Applicative, prefix terms
• Maximal subterm sharing (⇒ DAG)
  – cheap equality test, efficient rewriting
  – automatic generational garbage collection
• Annotations (text coordinates, dataflow info, ...)
• Concise, binary, sharing preserving encoding
• Language & machine independent
Why not using XML?

- Has been tried in various language processing projects
- XML is too verbose to represent parse trees of large (> 100 KLOC) programs
- XML does not provided sharing
- For discussion see: M.G.J. van Brand and P. Klint, ATerms for manipulation and exchange of structured data: It's all about sharing, *Information and Software Technology*, 49(1), 2007, 55-64.
The ToolBus architecture

Coordination

Representation

Computation

ATerms common data exchange format
The ToolBus architecture

- Processes inside the ToolBus can communicate with each other
- Tools can not communicate with each other
- Tools can communicate using fixed protocol:
A typical scenario

UI and DB are completely decoupled

Configuration knowledge only in ToolBus script

UI and DB are completely decoupled

User-interface  Database
The good news

• Communication between processes done via matching => no fixed channels, dynamic reconfiguration possible

• Large heterogenous applications have been built (> 300 KLOC) consisting of mixtures of C, Java, compiled ASF+SDF specs, ...

• In typical application: < 5% LOC is Tscript
The bad news

• Tools run as separate tasks at the OS level (this is standard in other SOAs)

• Tools and ToolBus communicate via TCP/IP
  => multi-processor applications possible
  => communication overhead on single processor

• Need for more flexibility, i.e. dynamic calls.

• The ToolBus implementation is small (10 KLoc) but complex; not so easy to experiment with
Roadmap

• Two examples of component-based systems
• Brief history of software components
• Coordination languages & SOA
• The ToolBus technology
• Next-generation ToolBus
  – regaining the costs of componentization
  – user-interface untangling
  – features under consideration
Roadmap

• Two examples of component-based systems
• Brief history of software components
• Coordination languages & SOA
• The ToolBus technology
• Next-generation ToolBus
  – regaining the costs of componentization
  – user-interface untangling
  – features under consideration
Current status (1)

• ToolBus has been reimplemented in Java
  – efficient, flexible, easily extensible
• Runs all existing test cases and most existing applications
• Some optimizations applied:
  – automata generation for process descriptions
  – static analysis of communication behaviour
    => flexible matching communication but efficiency of fixed channels
Current status (2)

• New features:
  – disrupt operator for error handling
  – property handling for easy configuration
  – dynamic loading of Java tools (see later)

• Features under considerations:
  – dynamic loading of Tscripts
  – optimization of tool-tool communication (see later)
Roadmap

- Two examples of component-based systems
- Brief history of software components
- Coordination languages & SOA
- The ToolBus technology
- Next-generation ToolBus
  - regaining the costs of componentization
  - user-interface untangling
  - features under consideration
Roadmap

- Two examples of component-based systems
- Brief history of software components
- Coordination languages & SOA
- The ToolBus technology
- Next-generation ToolBus
  - regaining the costs of componentization
  - user-interface untangling
  - features under consideration
Regaining costs componentization

Original system with tangled components: hard to maintain
Regaining costs componentization

Componentized system using ToolBus: very maintainable communication overhead
Regaining costs componentization

Dynamic loading of Java tools: keep components separate, but eliminate communication overhead
Observations

• Combination of components in arbitrary languages

• For Java components two choices:
  - dynamic loading
  - external execution with TCP/IP connection (possibly on other machine)

• Approach generalizes to components written in other languages (not yet implemented, but we understand how to handle C)
Roadmap

• Two examples of component-based systems
• Brief history of software components
• Coordination languages & SOA
• The ToolBus technology
• Next-generation ToolBus
  – regaining the costs of componentization
  – user-interface untangling
  – features under consideration
Roadmap

- Two examples of component-based systems
- Brief history of software components
- Coordination languages & SOA
- The ToolBus technology
- Next-generation ToolBus
  - regaining the costs of componentization
  - user-interface untangling
  - features under consideration
The ToolBus: a Service-oriented Architecture for language processing tools

User interface

User
The ToolBus: a Service-oriented Architecture for language processing tools

User

User interface

Intended connection between navigator tool and UI

Navigator

UI is a single tool; actual connection Fatter interface of UI tool

Paul Klint

The ToolBus: a Service-oriented Architecture for language processing tools
The ToolBus: a Service-oriented Architecture for language processing tools

User

User interface

Info tool

Fatter interface of UI tool
The ToolBus: a Service-oriented Architecture for language processing tools

Paul Klint

User interface

User

Graph display tool

Again: fatter interface of UI tool
Observations

- User interface is *undesired connection point*
- How to avoid that knowledge about components gets entangled in the UI?
- Solution:
  - UI becomes *plugin-architectuur*
  - plugin manages part of the UI
  - tools can *register a plugin*
  - tools broadcast user-interaction and also respond to it
The ToolBus: a Service-oriented Architecture for language processing tools

User

User interface

Paul Klint
The ToolBus: a Service-oriented Architecture for language processing tools

Paul Klint
The ToolBus: a Service-oriented Architecture for language processing tools

User interface

UI Plugins:
User interface

UI Plugins:

The ToolBus: a Service-oriented Architecture for language processing tools

Paul Klint
Roadmap

- Two examples of component-based systems
- Brief history of software components
- Coordination languages & SOA
- The ToolBus technology
- Next-generation ToolBus
  - regaining the costs of componentization
  - user-interface untangling
  - features under consideration
Roadmap

- Two examples of component-based systems
- Brief history of software components
- Coordination languages & SOA
- The ToolBus technology
- Next-generation ToolBus
  - regaining the costs of componentization
  - user-interface untangling
  - features under consideration
Features under consideration (1)

*tool-tool communication*

- Optimize tool-tool communication via direct data links between tools
- Needs (simple) distributed garbage collection
- Stream-based implementation

![Diagram showing tool-tool communication](image-url)
Features under consideration (2) *Streaming ATerms*

- Today (de)serialization of an ATrm requires an incore image of the serialized stream:

- A prototype is running that can send ATerms using a fixed size buffer:
Features under consideration (3) 

miscellaneous

• Dynamic loading of Tscripts
  – run-time customization of applications
  – distinguish: extensions and replacements
  – be aware of interference with running processes!

• Save/restore of (proces & data) of applications

• Transaction management
ToolBus: conclusions

- Separation of coordination and computation is very effective
- Workflow driven (~ web services)
- Truly component-based software
- Written in different programming languages
- Can execute as distributed system
- The components are reusable in many other applications
ToolBus: conclusions

• Proven in several applications (300-400KLOC)
• The next-generation ToolBus aims at:
  – decreasing the costs of componentization
  – eliminating communication bottlenecks
  – increasing run-time flexibility and extensibility
  – increasing reliability
Service-oriented Architecture

- Loose coupling
- Service contract
- Autonomy
- Abstraction
- Reusability
- Composability
- Statelessness
- Discoverability

- Message exchange patterns
- Coordination
- Atomic transactions
Key collaborators

Mark van den Brand   Hayco de Jong   Taeke Kooiker   Jurgen Vinju

Paul Klint   The ToolBus: a Service-oriented Architecture for language processing tools
Questions

See www.cwi.nl/~paulk for publications

See www.meta-environment.org for documentation and software downloads