Organizing and Evaluating Publish/Subscribe Systems with Scopes

Helge Parzyjegla, Matthias Prellwitz, Gero Mühl

Architecture of Application Systems
Department of Computer Science and Electrical Engineering
University of Rostock
Notification Service

$\text{publish}(n)$

$\text{subscribe}(F_n)$
Agenda

> Scoping
  > Scopes and interfaces
  > Scope hierarchies and visibility
  > Scope overlays and routing

> Management
  > Scope components
  > Joining a scope

> Evaluation
  > Scalability (number of brokers)
  > Disparate distributions

> Summary
  > Related Work
  > Conclusions
Scoping

> Fundamental concept for structured programming
  > Algol introduced code blocks (begin/end pairs) and nested function definitions

> Part of a computer program where a name binding is valid
  > Restricts the visibility of a program entity (e.g., a variable)

> Lexical / static scoping
  > Binding is valid within portion of the source code (area of text)
  > E.g., the definition of a function

> Dynamic scoping
  > Binding is valid within portion of run time
  > E.g., the execution of a function

```
int x = 1;
void foo() {
    printf("%i\n",x); x = 3;
}
void main() {
    int x = 2; foo(); printf("%i\n",x);
}
```

Static scoping in C outputs 1 and 2.

```
x=1
foo() { echo $x; x=3; }
main() { local x=2; foo; echo $x }
main
```

Dynamic scoping with Bash outputs 2 and 3.
Global visibility of published notifications

- Unrestricted communication between arbitrary system components
- Risk of unintended interferences grows with increasing system size
Scope Hierarchy

> Superscope/subscope
  > Induced by superset/subset relationship of components
  > Root scope contains all components
  > Components can be members of multiple scopes at the same time → intersecting scope sets
  > E.g., $C_3 \in R \cap T \subseteq S$

> Visibility of notifications
  > Induced by the interfaces of crossed scope boundaries
  > E.g., $C_1 \overset{n}{\rightarrow} C_3 \iff \operatorname{in}_S(n) \land \operatorname{in}_T(n)$
Scope Overlays

> Mapping of scope hierarchy into broker network → scope overlays
  > Overlay comprises scope’s components and connecting brokers
  > Brokers dynamically expand/shrink overlays based on demand
  > Routing has to consider scope boundaries and scope interfaces
Routing with Scopes

> Routing algorithms
  > Forwarding of message $m$ based on (routing) table entry $e$
  > Variants for notifications, subscriptions, and advertisements

> Case $S_m = S_e$ ($m$ and $e$ originate from same scope)
  > Conventional routing

> Case $S_m \neq S_e$ ($m$ and $e$ come from different scopes)
  > Superscope $S_{sup}$ of $m$ and $e$
  > Transformation $m \xrightarrow{T_{sup}} m'$ and $e \xrightarrow{T_{sup}} e'$ with $S_{m'} = S_{e'} = S_{sup}$

> Scoping and event routing are orthogonally composable features

Transformation algorithm for $m$

```plaintext
Δ ← $S_{sup} \setminus S_m$
while $Δ ≠ \emptyset \land m ≠ ε \land F_m ≠ false$
do
  let $S ∈ Δ$ with $∀ S' ∈ Δ : S' ⊂ S$
case $m$ of
    notification :
      $m ← out_S(m) ? m : ε$
    subscription :
      $F_m ← F_m ∧ in_S$
    advertisement :
      $F_m ← F_m ∧ out_S$
esac
  Δ ← Δ \setminus S
done
```
Management of Scope Overlays

> Separation of mechanism and policy
  > Brokers are able to deal with scoped messages and components as well as scope interface definitions → mechanism
  > Each scope is managed by a dedicated scope component responsible to provide a scope’s defining selector and interface filters as well as to approve scope memberships → policy

> Reuse of well-known publish/subscribe primitives and logic
  > Scope advertisements announce the existence of a particular scope disseminating the scope’s definition
  > Components issue scope subscriptions to request to join an advertised scope
  > Scope notifications inform about the approval or denial of a scope membership
Announcing Scopes

> Scope advertisements contain a scope’s definition (i.e., selector, interface filters, etc.)
> Disseminated/visible within overlay of its parent scope
Joining Scopes

> Scope subscriptions are routed on the reverse path of the advertisement towards the scope’s managing component

> May collect identity information of brokers on their way
Approving Scope Memberships

> Scope’s managing component approves join requests
> Approval notification is routed on the reverse path of the subscription expanding the scope’s overlay on its way
> Further subscriptions/advertisements may now become visible
Scope Attributes for Component Customization

> Scope Attributes

> Additional name/value pairs associated with a scope
> May be used for further context information about a scope
> Automatically added to notifications when published by components that are scope members

> Attribute inheritance

> Components as well as published notifications inherit attributes defined by superscopes
> Subscopes can overwrite inherited attribute values
> E.g., value ‘Germany’ of the attribute location can be set to ‘Cottbus’ by a more specific subscope
Evaluation

- Prototypical integration into Rebeca middleware
  - Native publish/subscribe implementation for small setups
- Discrete event simulation
  - Simulated network environment via PeerSim for larger setups
- Conducted experiments
  - Identity-based routing
  - With/without advertisements
  - With/without scopes
- Measured data
  - Routing table sizes
  - Control messages

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>10.000</td>
</tr>
<tr>
<td>Domänen</td>
<td>40</td>
</tr>
<tr>
<td>Broker</td>
<td>10…250</td>
</tr>
<tr>
<td>Publisher</td>
<td>100…2.500</td>
</tr>
<tr>
<td>Subscriber</td>
<td>100…2.500</td>
</tr>
<tr>
<td>Birthrate</td>
<td>0..10 s⁻¹</td>
</tr>
<tr>
<td>Lifetime</td>
<td>200 s</td>
</tr>
<tr>
<td>Notification types</td>
<td>250</td>
</tr>
<tr>
<td>Publication rate</td>
<td>1 s⁻¹</td>
</tr>
<tr>
<td>Scoping</td>
<td></td>
</tr>
<tr>
<td>Scopes</td>
<td>250</td>
</tr>
<tr>
<td>Hierarchy levels</td>
<td>3</td>
</tr>
<tr>
<td>Arity</td>
<td>2</td>
</tr>
</tbody>
</table>

Relevant simulation parameters.
Routing entries are only stored within the corresponding scope overlays → smaller publish/subscribe routing tables
Scoping reduces the number of control messages cutting of peak loads, but it cannot profit from saturation effects to the same extent.
Hot Spot Regions (Unequal Distributions)

Scoping effectively reduces the control traffic in case of regional concentrations of clients and interests.
Related Work

> Visibility control via scopes in event-based systems
  > Ludger Fiege explored the design space for scoping
  > Complex component graph to encode module structure, visibility constraints, and communication relationships
  > Multiple integration approaches without evaluations

> Security in (multi-domain) publish/subscribe systems
  > Role-based access control schemes to define which information is visible to whom
  > Focus on encryption techniques to ensure confidentiality

> Middleware standards and implementations
  > Manual visibility control by bridging/filtering between messaging domains (CORBA Notification Service, JMS, etc.)
  > AMQP supports flexible wiring of exchanges and queues
  > DDS provides communication planes as distinctive scopes
Conclusions

> Scopes in publish/subscribe restrict the visibility of notifications
  > Concise definition based on component sets
  > Can be imposed on existing components and infrastructures
  > Arbitrary complex scope hierarchies on flat broker network

> Scopes provide structuring means for organizing publish/subscribe systems and applications
  > Module concept for application developers
  > System organization for administrators
  > Component customization via scope attributes

> Qualitative and quantitative evaluation
  > Scopes reduce routing table sizes and respective update traffic
  > Limit negative effects of unbalanced and dynamic distributions
  > Provide an alternative to the usage of publisher advertisements
Thank you for your kind attention!

Dr.-Ing. Helge Parzyjegla
helge.parzyjegla@uni-rostock.de
http://wwwava.informatik.uni-rostock.de