

Managing Feature Interaction by Documenting and Enforcing Dependencies in Software Product Lines

Roberto S. Silva Filho and
David F. Redmiles

Donald Bren School of Information and Computer Sciences
Department of Informatics
University of California, Irvine
{rsilvafi, redmiles}@ics.uci.edu

International Conference on Feature Interaction, Grenoble, France, 3-5 Sep. 2007

Introduction

- This project was motivated by our research in the area of infrastructures for collaboration.
- We were interested in the design of a notification server that could support the different requirements from:
 - Collaborative software engineering
 - Software monitoring
 - P2P groupware tools
 - Other applications to come...
- For such, we designed and implemented YANCEES (Yet Another Configurable and Extensible Event Service)

Introduction (cont)

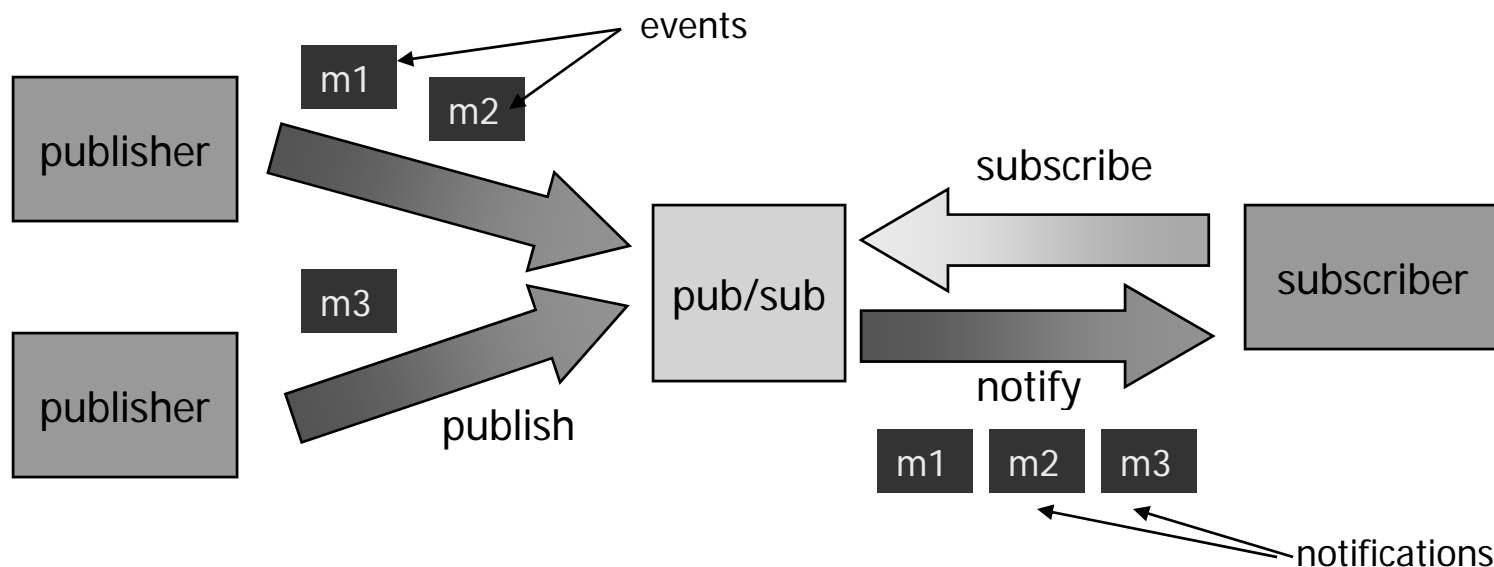
- In the design and implementation of YANCEES we adopted:
 - Well-known software product lines (SPL) methodology
 - and variability implementation approaches
- YANCEES was extended and configured in the support of different applications
 - P2P collaboration tools, application monitoring and awareness
- However, during the design, implementation and use of the infrastructure, many issues emerged
- The analysis of these issues revealed dependencies as the major factors behind these problems
- In particular, the lack of documentation and enforcement of dependencies may mislead software engineers in the extension and configuration of the SPL, leading to feature interference

Introduction (cont)

- The first part of the presentation discusses our experience in the design and implementation of YANCEES focusing on:
 - The kinds of feature interference faced in this process
 - The role of dependencies behind those issues
- In the second part, we discuss our current solution to feature interaction management:
- A formal notation to document dependencies in the product line code
 - That is both human-readable and
 - Automatically enforced by the infrastructure

Background: Publish/Subscribe infrastructures

- The publish/subscribe communication style provides:
 - Location and timing decoupling between producers and consumers of information
 - Supports 1-to-n event-based communication, with optional filtering mechanisms
- This communication style is usually implemented by a logically centralized service
 - That intermediates the communication between publishers and subscribers of events (or messages) in a distributed setting.



YANCEES case study

Yet ANother Configurable and
Extensible Event Service

YANCEES motivation

- Motivated by the need to support application domains with different requirements (or feature sets)
- For example:
 - Software monitoring
 - Subscription supporting event sequence detection, summarization and abstraction
 - Awareness applications
 - Protocols that allow event source browsing and advertisement
 - Peer-to-peer collaboration
 - Fast event routing with P2P federation of servers
 - and others...

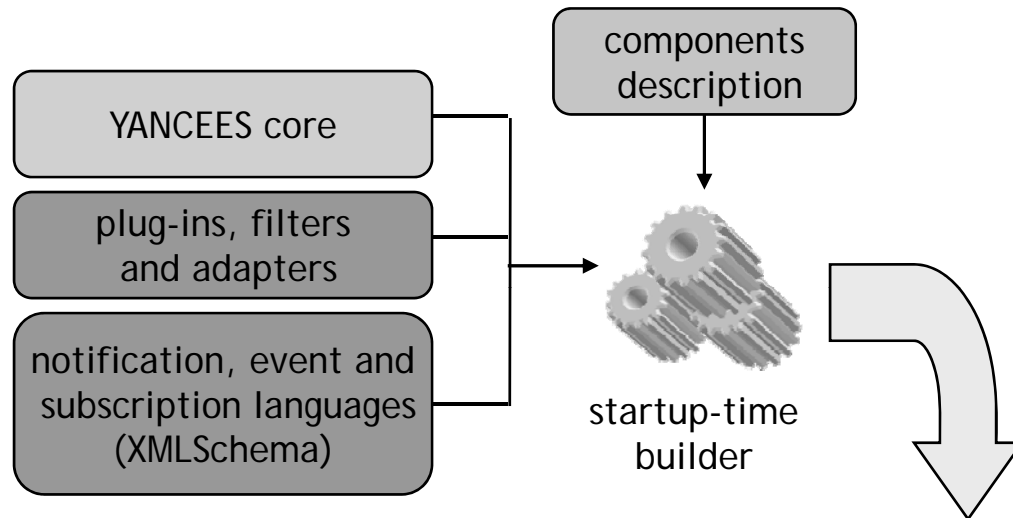
YANCEES commonality and variability design

- Common publish/subscribe behavior
- Designed support for different variability dimensions around an extended version of [Roseblum, Wolf 97] model
 - Event (record, object, attribute/value pair, text)
 - Subscription (sequence or content-based operators)
 - Notification (push, pull)
 - Protocol (infrastructure: P2P and user-level: mobility)
 - Publication (global filtering)
- Achieving different emerging characteristics
 - Event delivery guarantees (best effort, assured delivery)
 - Order (total order, partial order, best effort)
 - Timing (delays between consecutive events)

YANCEES Implementation strategies

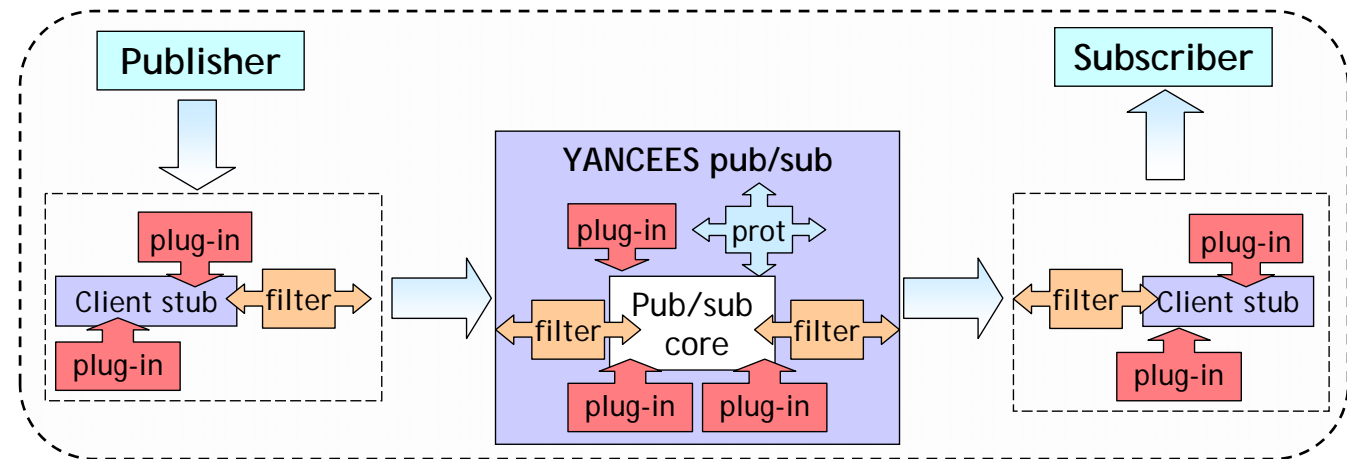
- Reuse of common pub/sub behavior:
 - Framework providing abstract publish/subscribe process
 - Generic event representations
- Runtime variability supporting dynamic subscriptions/notification policies through the use of:
 - Extensible languages (XML) and plug-ins
 - Controlled by dynamic parsers and builders
- Load-time variability of static services (protocol, publication, event format) by using:
 - Static general-purpose plug-ins
 - Adapters
 - Filters
 - Controlled by architecture manager (builder)

YANCEES Approach Summary



- Statically loaded filters and protocol plug-ins
- Dynamically loaded subscription and notification plug-ins
 - according to user-provided subscriptions

domain-specific YANCEES instance



YANCEES Design and Implementation Issues

The role of dependencies in limiting configurability and extensibility in software

The role of dependencies

- Fundamental (or problem domain) dependencies
 - Integrate the common components of the infrastructure
- Configuration-specific dependencies
 - Integrate features that extend the common behavior
- Incidental (or technological) dependencies
 - Come as a consequence of the implementation and variability realization approaches adopted.
- Implicit dependencies on emerging system properties
 - Represent complex behavior dependencies in the system

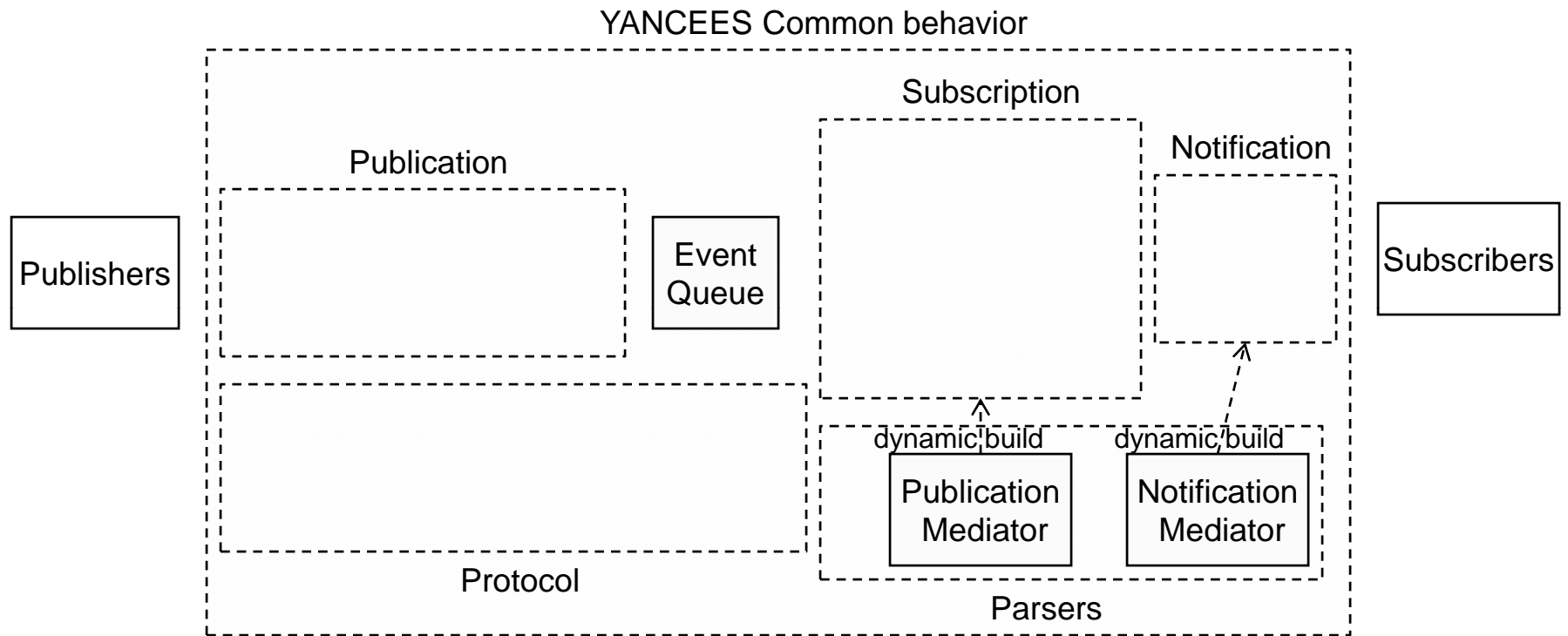
YANCEES implementation issues

- Fundamental dependencies. The variability dimensions selected were not orthogonal
 - For example, changes in the event format, federation protocol, or routing strategy may impact features in other variation points.
- Configuration-specific dependencies. Changes in existing features may impact other features that depend on them
- Incidental dependencies. The benefits of using variability approaches (design patterns, plug-ins, extensible languages, etc.) come with extra costs:
 - new configuration rules
 - Including activation and installation order, incompatibilities, and use dependencies
- Emerging System properties. Developers cannot understand the global consequences of choosing a complex set of features
 - For example, the impact of changes in the timing and event delivery guarantees

Dependencies in action

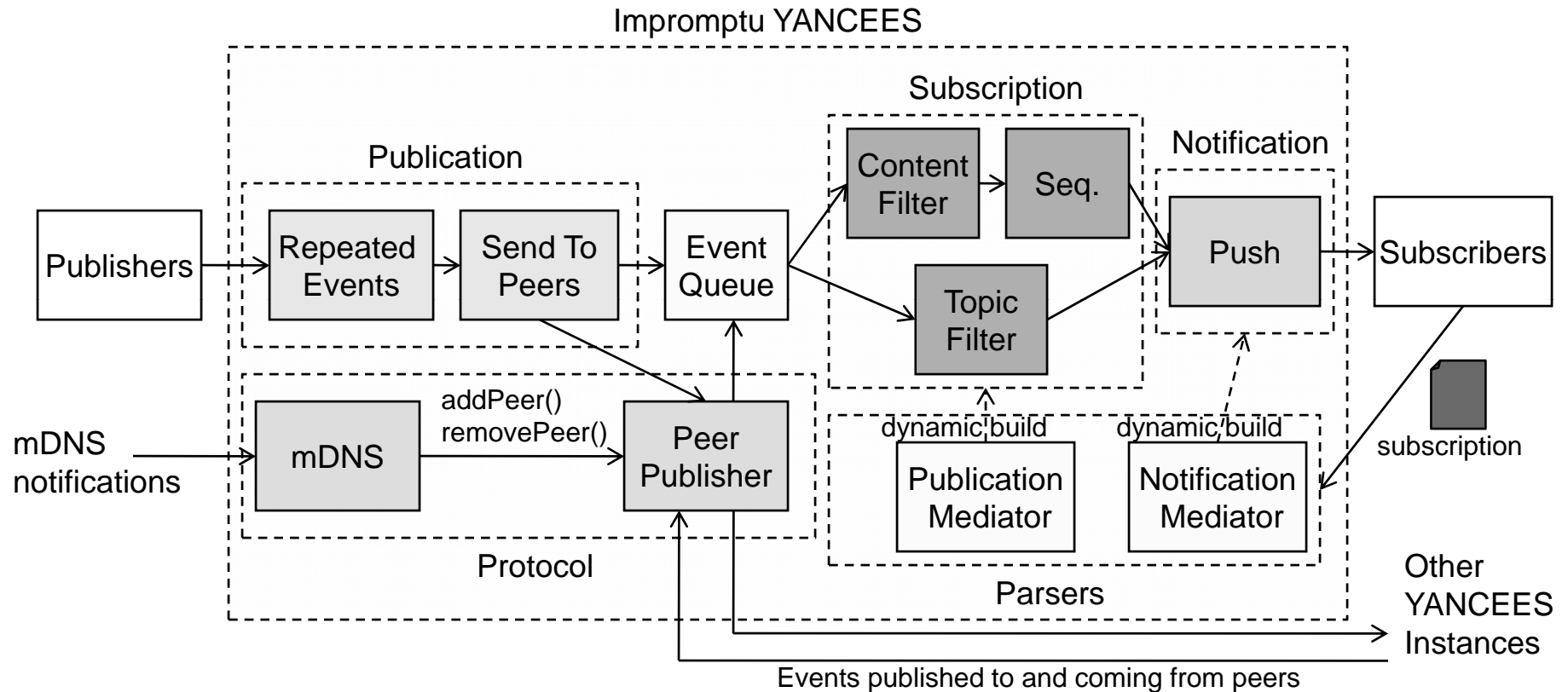
Example: extending YANCEES to
support Impromptu, a P2P file
sharing tool

YANCEES Generalized framework



- Dynamic parsers (publication and notification mediators) allocate subscription and notification plug-ins on demand
- Other variation points: protocol and publication models are extended through the use of static plug-ins

Configuration Example

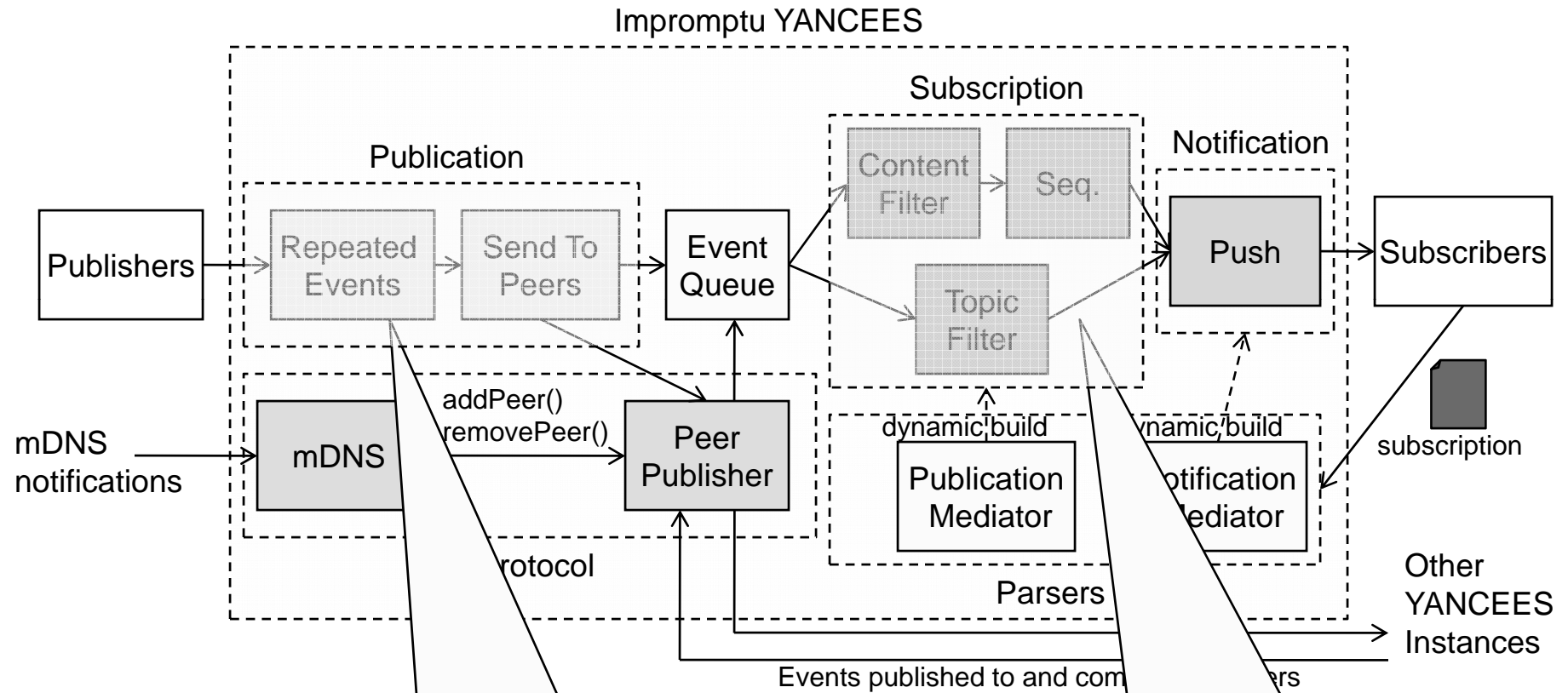


- Application: Impromptu P2P file sharing
- Configuration
 - Event : Attribute/value
 - Subscription: Content or topic filter with sequence detection
 - Protocol: P2P → multicast DNS and publication interceptor
 - Notification: Push
 - Publication: repeated events filtering

Dependency-driven interference

Extensibility and configurability
issues in YANCEES

Fundamental (or problem domain) dependencies

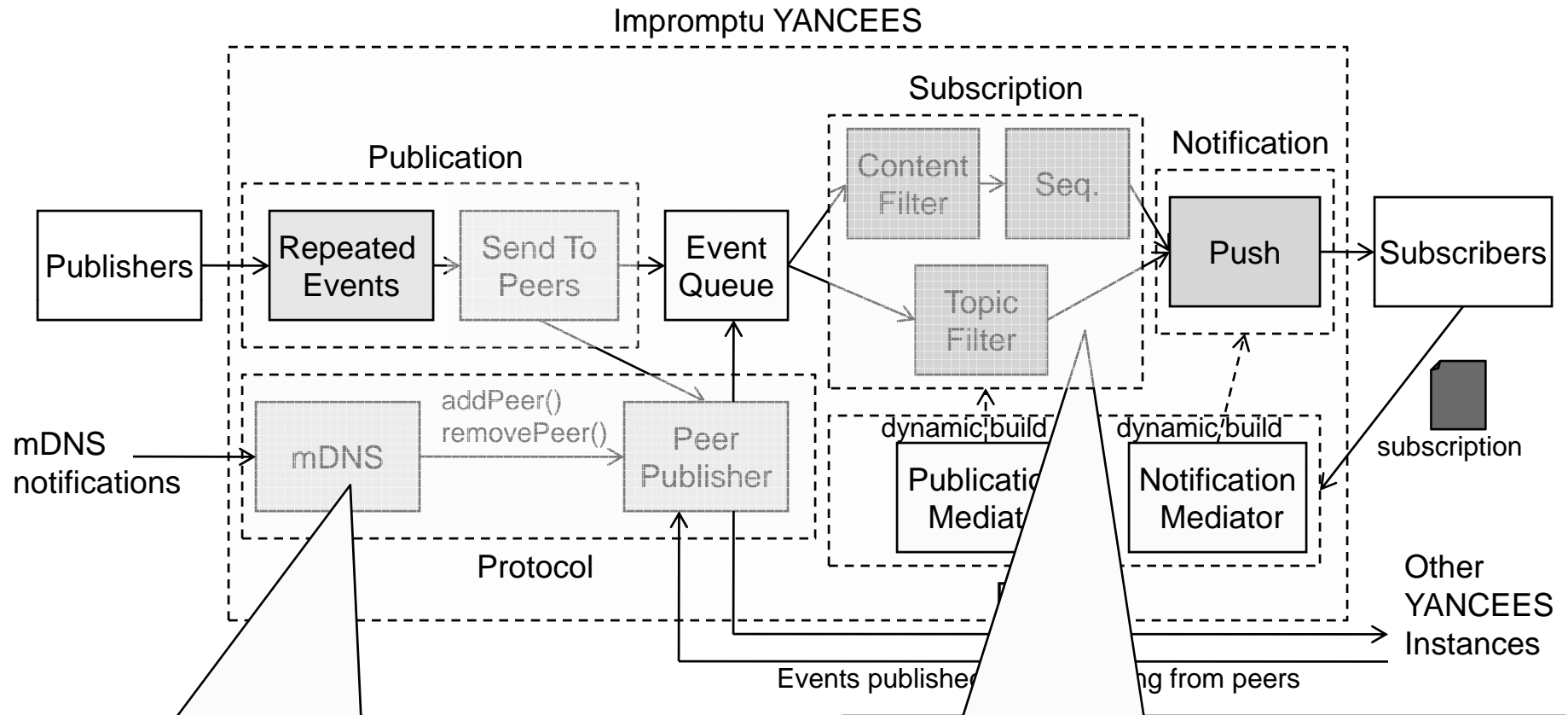


Through fundamental dependencies, changes in the event format, timing and order behaviors may:

- invalidate the publication filters

- It can also invalidate subscription plug-ins

Configuration-specific dependencies

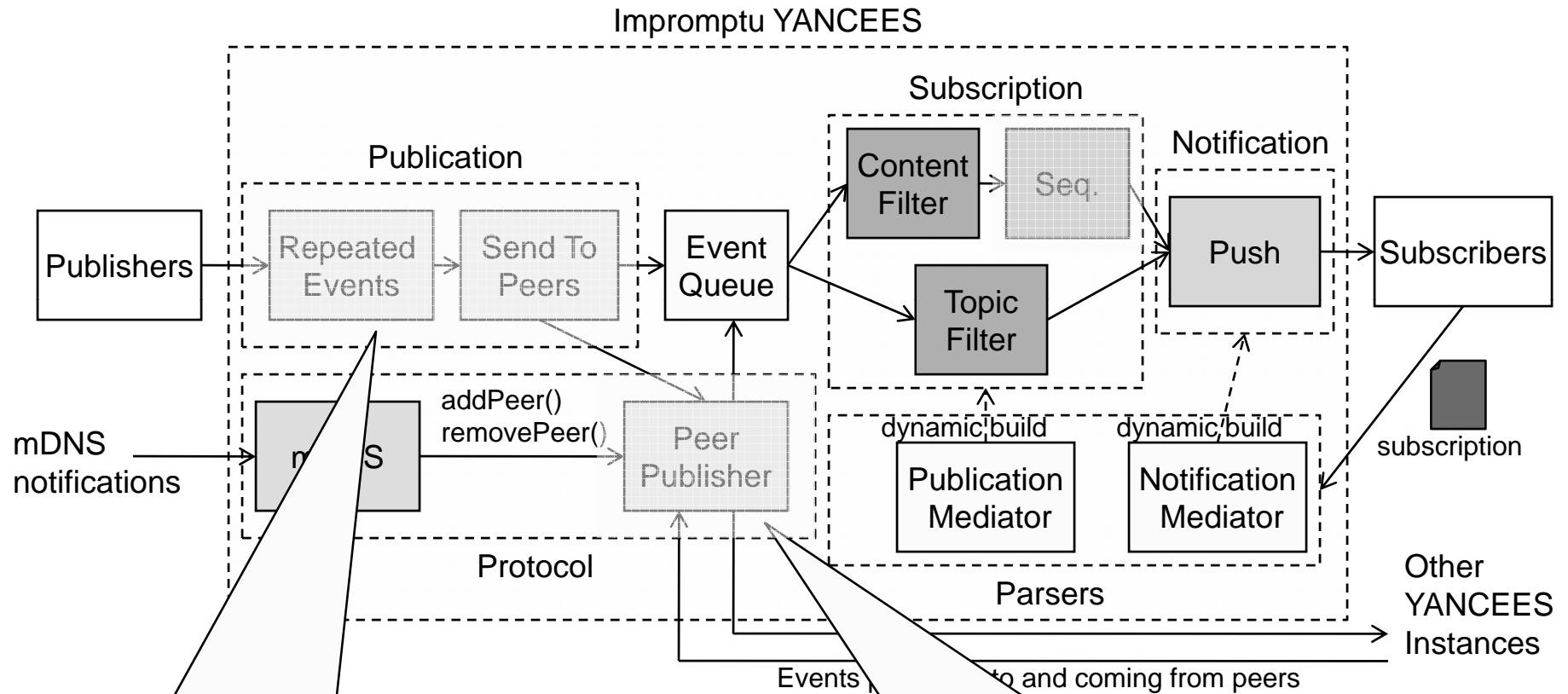


Through configuration-specific dependencies:

- the P2P publication will only work if all the components are properly installed
- Incompatible versions may result in interference

- sequence detection plug-in works with *ContentFilter* but not with *Topic* filter
- topic and content filters may change the order of the events when taking part in the same subscription

Incidental (or technological) dependencies

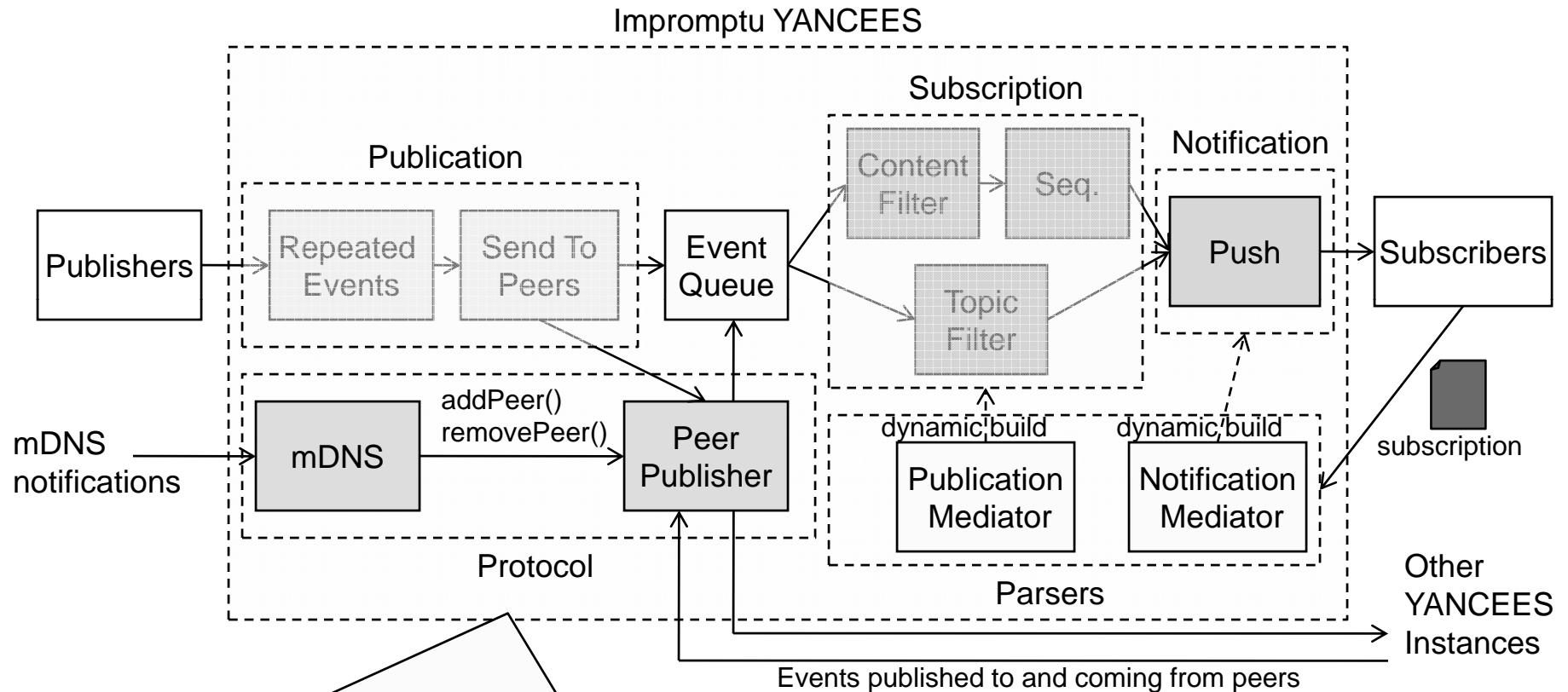


the chain-of responsibility may lead to interference if changes occur in:

- the order of the filters
- the content or number of attributes in the events

Multiple writes queue: through *PeerPublisher*, events may come out of order when examined by the subscription filters

Dependencies on emerging system properties



Implicit assumptions exist with respect to emerging system properties. These properties may change due to complex dependencies:

- Timing (delay between events) → distribution, input filters side effects
- Order of the events (total, partial) → input filters, peer publisher plug-in

Managing dependency-driven interference

By documenting and enforcing dependencies in the code

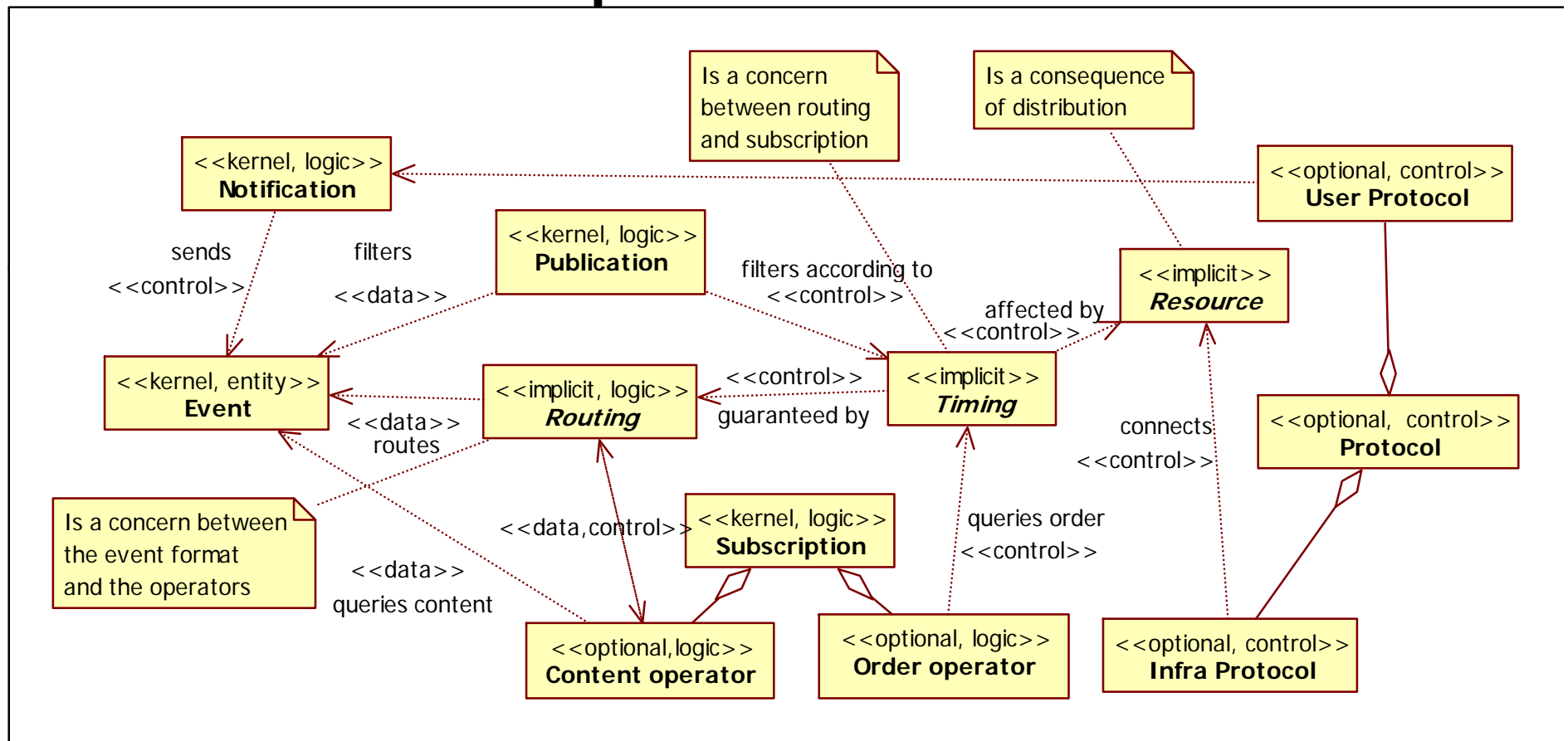
Approach

- Document and enforce different kinds of dependencies in the infrastructure.
 - Make dependencies explicit since the design
 - Use dependency analysis models
 - Make dependencies explicit in the code
 - At both variation points and feature implementation
 - Enforce these properties through the use of static and dynamic configuration managers
 - Runtime subscription/notification parsers and composition filters
 - Load time static configuration manager (architecture manager)

Making dependencies explicit in the design

Using additional dependency
analysis models

Analysis of fundamental dependencies



- In the product line design, consider the impact of fundamental dependencies
- Also model implicit properties and configuration-specific dependencies (in italic)

Implementation

- YANCEES was extended with:
 - Documentation notation:
 - Based on JDK 1.6 annotations API, expressing the different kinds of dependencies
 - A variability model (*VariabilityModel.java*) that provides a single point of access to the main variation points and emerging properties
 - Enforcement capability:
 - Dependency-aware dynamic and static parsers
 - Composition filters (implemented as plug-ins and filers wrappers) to enforce provided and required configuration-specific properties at runtime

Code Example – input filter and variation point annotations

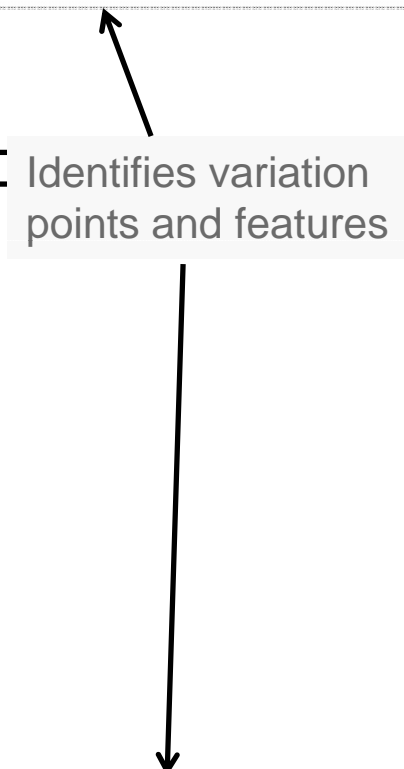
```
// Abstract input filter Variation Point  
  
public abstract class AbstractFilter implements FilterInterface {  
    //--- Abstract implementation goes here ---  
}
```

```
// concrete implementation of the PublishToPeers filter Feature  
  
public class SentToPeersInputFilter extends AbstractFilter {  
    // --- plug-in implementation ---  
}
```

Code Example – input filter and variation point annotations

```
// --- Indicates what variation point this class implements ---  
@ImplementsVariationPoint(VariabilityModel.VariationPoints.PUBLICATION)  
  
public abstract class AbstractFilter implements FilterInterface {  
    //--- Abstract implementation goes here ---  
}
```

Identifies variation
points and features



```
// --- Feature unique ID ---  
@ImplementsFeature(name = "Publication.PublishToPeers", version="1.0")  
public class SentToPeersInputFilter extends AbstractFilter {  
    // --- plug-in implementation ---  
}
```

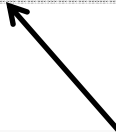
Code Example – fundamental dependencies

```
//--- Indicates fundamental dependencies on other variation points ---  
@DependsOnVP(VariabilityModel.VariationPoints.EVENT)
```

```
// --- Indicates what variation point this class implements ---  
@ImplementsVariationPoint(VariabilityModel.VariationPoints.PUBLICATION)
```

```
public abstract class AbstractFilter implements FilterInterface {  
    //--- Abstract implementation goes here ---  
}
```

Fundamental
dependency to
the event model



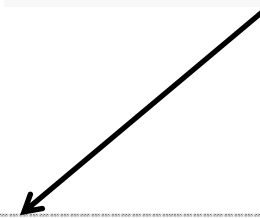
```
// --- Feature unique ID ---  
@ImplementsFeature(name = "Publication.PublishToPeers", version="1.0")  
public class SentToPeersInputFilter extends AbstractFilter {  
    // --- plug-in implementation ---  
}
```

Code Example – configuration-specific dependencies

```
//--- Indicates fundamental dependencies on other variation points ---  
@DependsOnVP(VariabilityModel.VariationPoints.EVENT)  
  
// --- Indicates what variation point this class implements ---  
@ImplementsVariationPoint(VariabilityModel.VariationPoints.PUBLICATION)
```

```
public abstract class AbstractFilter implements FilterInterface {  
    //--- Abstract implementation goes here ---  
}
```

Configuration-specific
dependency to:
YanceesEvent.class
implementation



```
// --- Compatibility with features and emerging properties ---  
@CompatibleWithFeature(  
    variationPontType = VariabilityModel.VariationPoints.EVENT,  
    featureClass= edu.uci.isr.yancees.YanceesEvent.class, version="3.0",  
    featureName="Event.AttributeValueEvent")
```

```
// --- Feature unique ID ---  
@ImplementsFeature(name = "Publication.PublishToPeers", version="1.0")  
public class SentToPeersInputFilter extends AbstractFilter {  
    // --- plug-in implementation ---  
}
```

Code Example – incidental dependencies


```
//--- Indicates fundamental dependencies on other variation points ---  
@DependsOnVP(VariabilityModel.VariationPoints.EVENT)  
  
// --- Indicates what variation point this class implements ---  
@ImplementsVariationPoint(VariabilityModel.VariationPoints.PUBLICATION)  
  
public abstract class AbstractFilter implements FilterInterface {  
    //--- Abstract implementation goes here ---  
}
```

```
// --- Local configuration concerns ---  
@ProvidedGuarantees(modifyEventContent=false, modifyEventOrder=false,  
    modifyEventType=false)  
@RequiredGuarantees(intactEventContent=false, intactEventOrder=false,  
    intactEventType=false)
```

```
// --- Compatibility with features and emerging properties ---  
@CompatibleWithFeature(  
    variationPontType = VariabilityModel.VariationPoints.EVENT,  
    featureClass= edu.uci.isr.yancees.YanceesEvent.class,  
    featureName="Event.AttributeValueEvent")
```

```
// --- Feature unique ID ---  
@ImplementsFeature(name = "Publication.PublishToPeers", version="1.0")  
public class SentToPeersInputFilter extends AbstractFilter {  
    // --- plug-in implementation ---  
}
```

Incidental
dependencies tailored
to the chain of
responsibility pattern

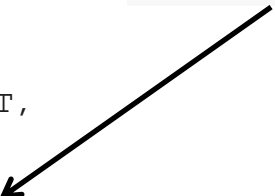


Code Example – emerging system properties

```
//--- Indicates fundamental dependencies on other variation points ---  
@DependsOnVP(VariabilityModel.VariationPoints.EVENT)  
  
// --- Indicates what variation point this class implements ---  
@ImplementsVariationPoint(VariabilityModel.VariationPoints.PUBLICATION)  
  
public abstract class AbstractFilter implements FilterInterface {  
    //--- Abstract implementation goes here ---  
}
```

```
// --- Local configuration concerns ---  
@ProvidedGuarantees(modifyEventContent=false, modifyEventOrder=false,  
    modifyEventType=false)  
@RequiredGuarantees(intactEventContent=false, intactEventOrder=false,  
    intactEventType=false)  
  
// --- Compatibility with features and emerging properties ---  
@CompatibleWithFeature(  
    variationPontType = VariabilityModel.VariationPoints.EVENT,  
    featureClass= edu.uci.isr.yancees.YanceesEvent.class,  
    featureName="Event.AttributeValueEvent")  
@CompatibleWithProperties(  
    resource = VariabilityModel.Resource.ANY,  
    routing = VariabilityModel.Routing.ANY,  
    timing = VariabilityModel.Timing.ANY)
```

Emerging system
properties
compatibility



```
// --- Feature unique ID ---  
@ImplementsFeature(name = "Publication.PublishToPeers", version="1.0")  
public class SentToPeersInputFilter extends AbstractFilter {  
    // --- plug-in implementation ---  
}
```


Outcome

- By using this strategy, software product line developers can:
 - Document the existing dependencies
 - Provide mechanisms to enforce the different dependencies
- Whereas software engineers that extend and configure the infrastructure are informed about inconsistencies
 - At extension time, when reading the code and programming the extension
 - At configuration time, when deciding which components to integrate
 - At load time through error messages
 - At runtime through exceptions (on subscription and publication commands)

Conclusions

- The gains in reuse and variability provided by SPLs as YANCEES come with the increase in the software complexity
- This complexity is a function of different kinds of software dependencies and the design for variability
- When not documented and managed, dependencies lead to feature interference.
- This paper:
 - Describes the kinds of dependencies found in the design and implementation of YANCEES
 - And exemplifies the feature interference resulting from these dependencies
 - Proposes a formal documentation strategy to make dependencies explicit in a way that:
 - Is both machine and human readable
 - Support runtime and static configuration
 - Helping software engineers in preventing feature interference

Future Work

- Automate the detection of dependencies
 - Currently, they are manually annotated by software engineers
 - And provided as template code for third-party developers, at every variation point
- Perform user studies to determine the usability of the approach
- Generalize the annotation model and the framework, retrofitting existing frameworks/product lines
 - Currently, the annotations and implementation are tailored to support YANCEES only.

Thank you

Questions/comments?

References

- Bergmans, L. and Aksit, M. Composing Crosscutting Concerns Using Composition Filters. *Communications of the ACM*, 44 (10). 51-58.
- Birsan, D. On Plug-ins and Extensible Architectures *ACM Queue*, 2005, 40-46.
- Bosch, J., Evolution and Composition of Reusable Assets in Product-Line Architectures: A Case Study. in *TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, (1999), Kluwer, B.V, 321 - 340.
- Bowen, T.F., Dworack, F.S., Chow, C.H., Griffeth, N., Herman, G.E. and Lin, Y.-J., The feature interaction problem in telecommunications systems. in *Software Engineering for Telecommunication Switching Systems*, (1989), 59 - 62.
- Bryant, A., Catton, A., Volder, K.D. and Murphy, G.C., Explicit Programming. in *1st AOSD*, (Enschede, The Netherlands, 2002).
- Coplien, J., Hoffman, D. and Weiss, D. Commonality and Variability in Software Engineering *IEEE Software*, 1998, 37-45.
- Czarnecki, K. and Eisenecker, U.W. *Generative Programming - Methods, Tools, and Applications*. Addison-Wesley, 2000.
- Deelstra, S., Sinnema, M., Nijhuis, J. and Bosch, J. Experiences in Software Product Families: Problems and Issues during Product Derivation, *SPLC'04. Springer Verlag LNCS*, 3154. 165-182.
- DePaula, R., Ding, X., Dourish, P., Nies, K., Pillet, B., Redmiles, D., Ren, J., Rode, J. and Silva Filho, R.S. In the Eye of the Beholder: A Visualization-based Approach to Information System Security. *IJCHS - Special Issue on HCI Research in Privacy and Security*, 63 (1-2). 5-24.
- Dingel, J., Garlan, D., Jha, S. and Notkin, D., Reasoning about implicit invocation. in *6th International Symposium on the Foundations of Software Engineering (FSE-6)*, (Lake Buena Vista, FL, USA, 1998).
- Ferber, S., Haag, J. and Savolainen, J. Feature Interaction and Dependencies: Modeling Features for Reengineering a Legacy Product Line. *LNCS. Second International Conference on Software Product Lines, SPLC'02*, 2379. 235-256.
- Fowler, M. Inversion of Control Containers and the Dependency Injection Pattern, <http://www.martinfowler.com/articles/injection.html>, 2004.
- Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company, 1995.
- Hunleth, F. and Cytron, R.K., Footprint and feature management using aspect-oriented programming techniques. in *Joint conference on Languages, compilers and tools for embedded systems*, (Berlin, Germany, 2002), ACM Press, 38 - 45.
- Jacobson, I., Griss, M. and Jonsson, P. *Software Reuse. Architecture, Process and Organization for Business Success*. Addison-Wesley, 1997.

References (cont)

- Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E. and Peterson, A.S. Feature-Oriented Domain Analysis (FODA) Feasibility Study - CMU/SEI-90-TR-021, Carnegie Mellon Software Engineering Institute, Pittsburgh, PA, 1990.
- Krueger, C. Software Product Line Concepts: www.softwareproductlines.com/introduction/concepts.html, The Software Product Lines site, 2006.
- Krueger, C.W., Software product line reuse in practice. in *3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, (Richardson, TX, USA, 2000), 117-118.
- Lee, K. and Kang, K.C. Feature Dependency Analysis for Product Line Component Design. *Lecture Notes in Computer Science - 8th International Conference on Software Reuse, ICSR'04*, 3107. 69-85.
- Metzger, A., Bühne, S., Lauenroth, K. and Pohl, K., Considering Feature Interactions in Product Lines: Towards the Automatic Derivation of Dependencies between Product Variants. in *Feature Interactions in Telecommunications and Software Systems VIII*, (Leicester, UK, 2005), 198-216.
- Rosenblum, D.S. and Wolf, A.L., A Design Framework for Internet-Scale Event Observation and Notification. in *6th ESEC/FSE*, (Zurich, 1997), Springer-Verlag, 344-360.
- Silva Filho, R.S. and Redmiles, D., Striving for Versatility in Publish/Subscribe Infrastructures. in *5th International Workshop on Software Engineering and Middleware (SEM'2005)*, (Lisbon, Portugal., 2005), ACM Press, 17 - 24.
- Silva Filho, R.S. and Redmiles, D.F. A Survey on Versatility for Publish/Subscribe Infrastructures. Technical Report UCI-ISR-05-8, ISR, Irvine, CA, 2005, 1-77.
- Sinnema, M. and Deelstra, S. Classifying variability modeling techniques. *Information and Software Technology*, 49 (7). 717-739.
- Sinnema, M., Deelstra, S., Nijhuis, J. and Bosch, J. COVAMOF: A Framework for Modeling Variability in Software Product Families. *LNCS*, 3154/2004. 197-213.
- Svahnberg, M., Gurf, J.v. and Bosch, J. A Taxonomy of Variability Realization Techniques. *Software Practice and Experience*, 35 (8). 705-754.
- Szyperski, C. *Component Software: Beyond Object-Oriented Programming, 2nd edition*. ACM Press, 2002.
- Zibman, I., Woolf, C., O'Reilly, P., Strickland, L., Willis, D. and Visser, J. An architectural approach to minimizing feature interactions in telecommunications. *IEEE/ACM Transactions on Networking*, 4 (4). 582-596.