The Design of a Configurable, Extensible and Dynamic Notification Service

Roberto S. Silva Filho

Cleidson R. B. de Souza

David F. Redmiles

School of Information and Computer Science

UCI - University of California, Irvine

{rsilvafi, cdesouza, redmiles}@ics.uci.edu

Outline

- Motivation Problem
- Approach
- Design
- Examples
- Implementation
- Conclusions and Future work

Project Motivation

- The need for an event-based infrastructure to:
  - support requirements from different application domains
    - groupware, software monitoring, awareness, mobility...
  - support new functionality as necessary
  - provide the right functionality set to each application domain
  - provide a single model for different applications

Application domains

- What we wanted is a configurable event-notification service that can be easily customized, and extensible to support different domains such as:
- Mobility
  - pull, persistency, roaming protocol, authentication
- Awareness
  - event persistency and typing, event validity (time-to-live), event sequence detection, push and pull delivery; event source browsing (discovery)
- Application monitoring
  - event sequence detection; event abstraction; browsing of information sources and their events; event persistency; push and pull

Problems with current event notification servers

- Specialized approaches
  - Domain specific notification servers
    - such as Khronika, CASSIUS, JEDI, EBBA
- Generic approaches
  - “one-size-fits-all”
    - such as READY, CORBA-NS
  - content-based
    - such as Siena, Elvin
- Problem: poor or no support for extensibility and configurability

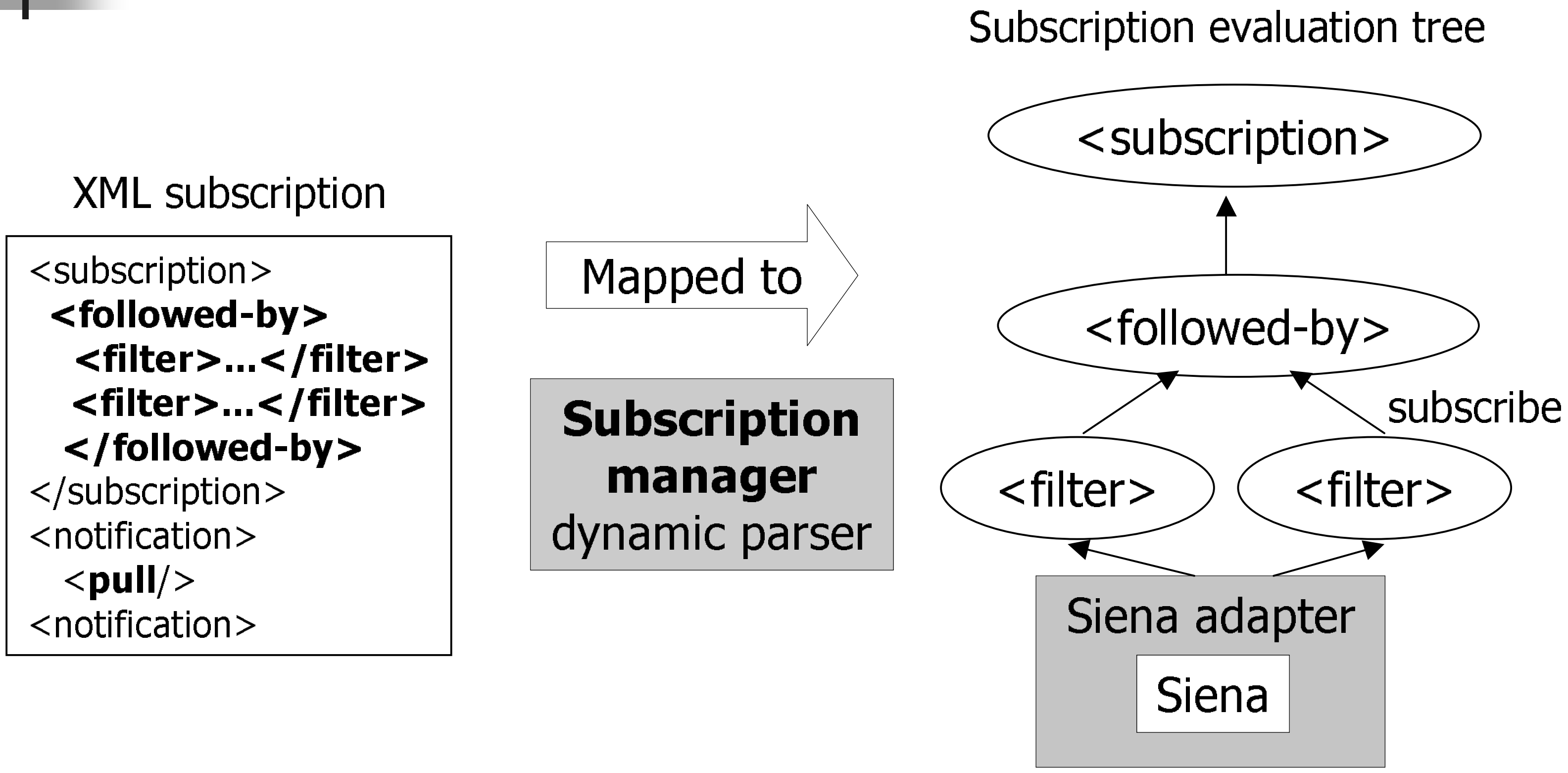
Our Approach

- Provide a framework to support extensibility and configurability of notification servers
- Based on:
  - Plug-ins
  - Extensible event, notification and subscription languages
  - Extensible protocols
  - Dynamic parsers
  - Configuration managers
  - Around a simple publish/subscribe core

Our approach

- Configurations are represented as sets of plug-ins and a publish/subscribe core adapter
- Plug-ins are used to extend the basic event dispatcher functionality, notification mechanisms and protocols
- Parsers convert **subscriptions, notification preferences** and **protocols** into evaluation trees based on plug-in instances
- Plug-ins can be downloaded, at runtime, if not currently installed

Adapter extension using plug-ins



- Approach valid to protocol, notification and protocol plug-ins too

Our strategy

- To address the problem based on the design models proposed by [*Cugola et al. 01*] and inspired by [*Rosemblum and Wolf 97*].
- In other words, provide a way to customize and extend the following design models:
  - Event
  - Subscription
  - Notification
  - Resource
  - Protocol (introduced here)

Notification, Subscription and Protocol Models

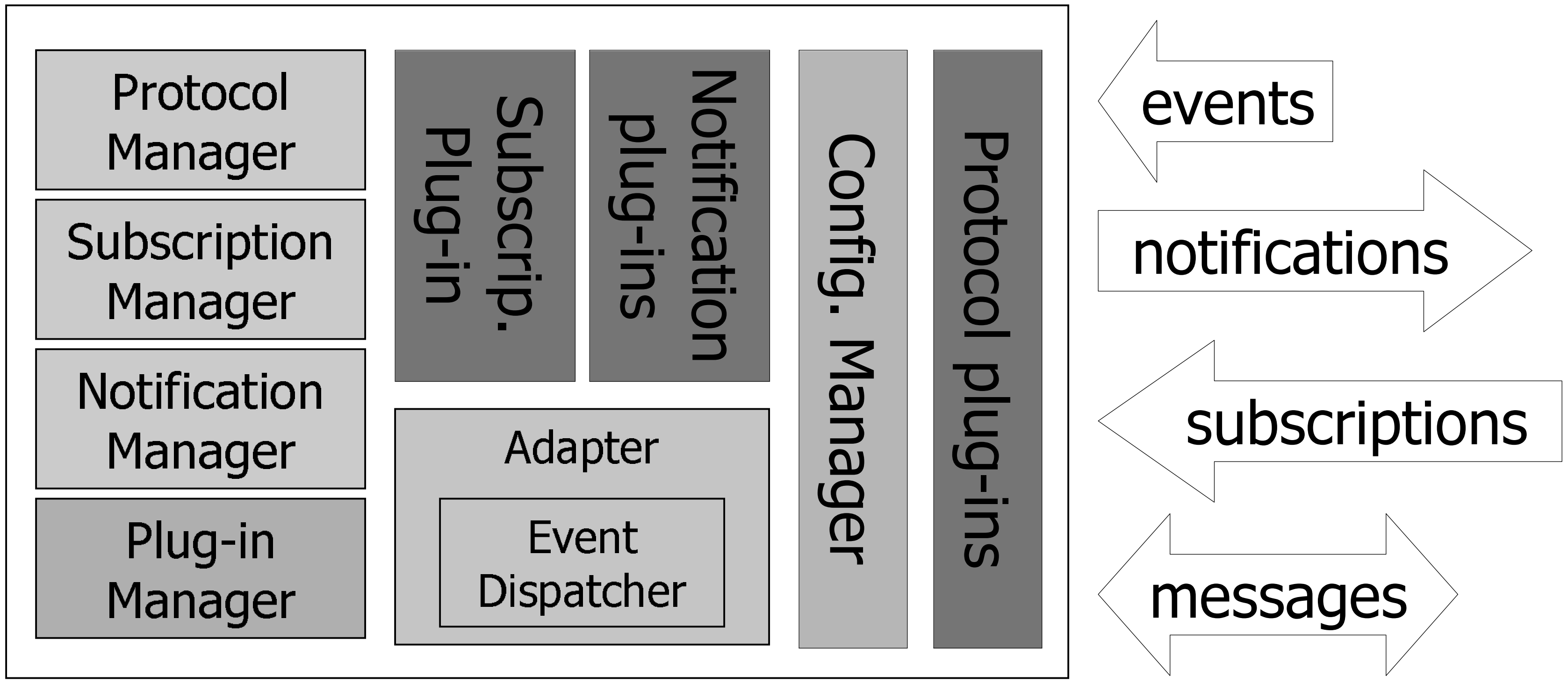
- **Event model**
  - Example: Tuple-based, type-based, object-based
- **Subscription model**
  - Example: sequence, abstraction, rules, content-based queries, and so on…
- **Notification model**
  - Example: push, pull, other notification policy…

Event and Resource models

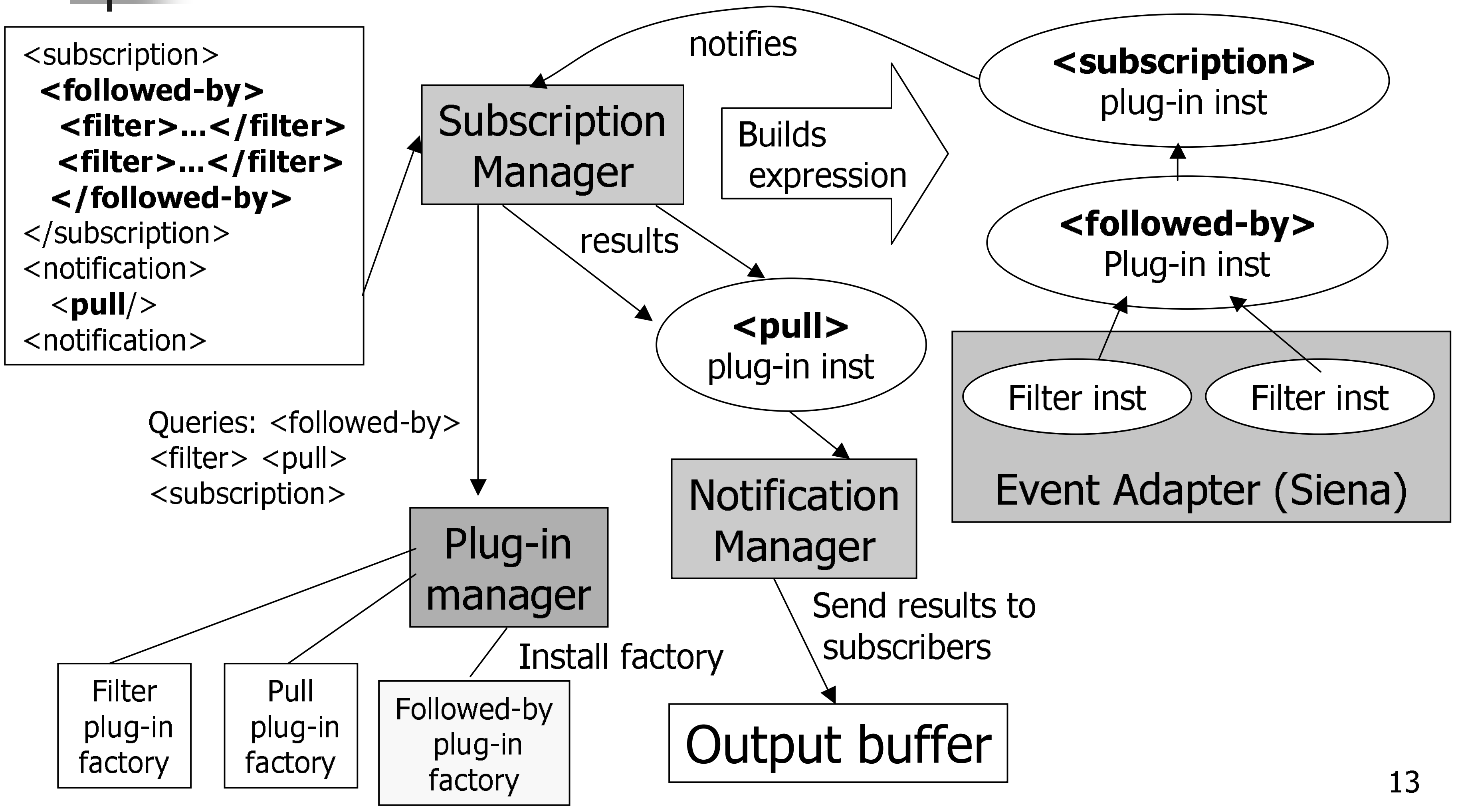
- **Resource model**
  - Example: client side and server side plug-ins
- **Protocol model**
  - Example: security, mobility, authentication…

- All models are **extended** by:
  - Plug-ins
  - Specific language definitions
  - Managers that interpret the language with the plug-ins.

Architecture overview



Subscription parsing example



Extensibility summary

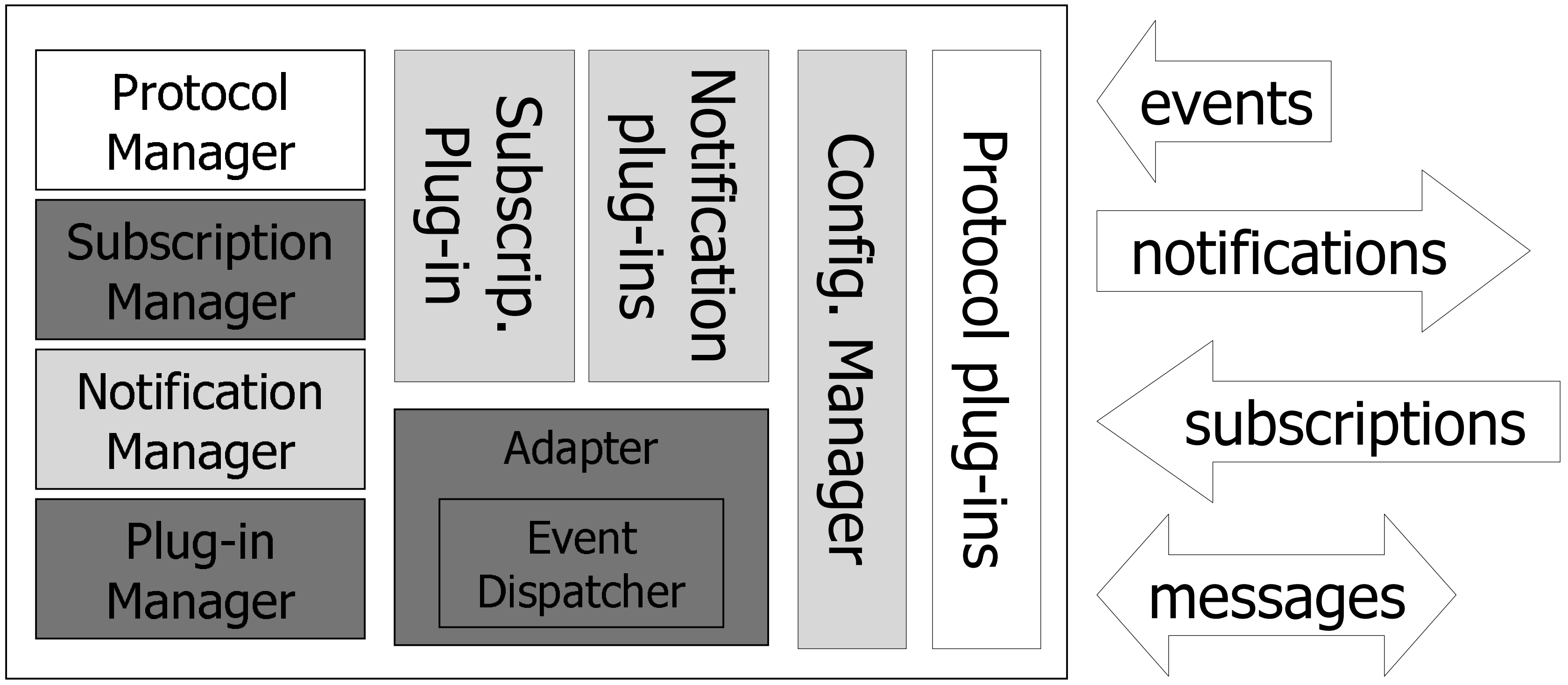
DESIGN DIMENSION	HOW TO EXTEND	EXAMPLES
Subscription Model	Extensible **subscription language** Provide feature specific event processing **plug-ins**	Event aggregation Abstraction Sequence detection
Event Model	Extensible **event representation language** An **event adapter** for each dispatcher used **Plug-in** to handle the dispatcher specific event language	Tuple based Record based Object based
Notification Model	Notification **plug-ins** (or filters) Extensible **notification language** that allows the definition of notification policies	Push Pull (with persistency)
Resource Model	Server **configuration language** and **configuration manager** that allows the distribution of event processing to server-side or client-side plug-ins	Centralized Partially distributed
Protocol Model	Extensible **protocol language** Protocol **plug-ins** and **protocol manager** to handle different protocols	Security protocols Mobility protocols Configuration protocols

Implementation Status

- The following components are implemented:
  - Subscription manager
  - Plug-in manager
  - Event dispatcher adapter using Siena.
  - Simple plug-ins: sequence detection, rules

- The other components will be ready by the end of summer

Implementation status



Conclusions

- Extensibility needs to address issues in all the models (notification, subscription, event, resource) discussed. This can be addressed by:
    - Runtime composition of plug-in instances
    - Extensible languages
    - Adapters (event dispatcher model)

- Plug-ins can also be used to better distribute processing through the components of the system.

Conclusions

- Configurability is provided by:
  - The installation of specific plug-ins
  - Selection of plug-ins in a configuration language
- Dynamism:
  - Result of dynamic expression building
  - Implemented by the installation of plug-ins at runtime.

Future work

- Investigate the problems related to timing
- Improve the implementation
- Test by implementing different configurations
- Compare results with existing notification servers such as CASSIUS and CORBA-NS

- Analyze the benefits and weaknesses of this approach

Questions?

- Research group: awareness.ics.uci.edu
- Project: www.ics.uci.edu/~rsilvafi

References

- G. Cugola, E. D. Nitto, and A. Fuggeta, "The Jedi Event-Based Infrastructure and Its Application on the Development of the OPSS WFMS," IEEE Transactions on Software Engineering, vol. 27, pp. 827-849, 2001.

- D. S. Rosenblum and A. L. Wolf, "A Design Framework for Internet-Scale Event Observation and Notification," presented at 6th European Software Engineering Conference/5th ACM SIGSOFT Symposium on the Foundations of Software Engineering, Zurich, Switzerland, 1997.