Topic Proposal Revisited

OVERVIEW (after talking to Crista)

1) Problem characterization
Adopt a design model such as Rosemblum and Wolf in order to understand the main dimensions of a pub/sub infrastructure and their dependencies. A model is a small scale representation of the actual system that can be used to test it, predicting its behavior through generalization. The model here is also generic and complete enough to represent the possible variations of systems of such kind. This model is composed of different design dimensions and their inter-dependencies. In other words, this model characterizes the pub/sub problem in terms of its variability, and the influence of one dimension in another.

Moreover, because those dimensions are coupled (dependent), a variation in one dimension requires adjustments in the way the other dimension is implemented. Therefore, there is a need for an open implementation, a re-evaluation of strategies in one module, driven by the use of one variation in another module. Hence, variability results in crosscutting concerns throughout the design of a pub-sub infrastructure.

The whole idea of modularization used in structured design (Stevens, Myers et al. 1999) and more recently in OO design (Eder, Kappel et al. 1992), is the application of the principle of information hiding (Parnas 1972) to insulate changes behind well-defined interfaces. When we want to create product-line architectures, however, this insulation is not always possible. In fact, the idea of open implementation (Maeda, Lee et al. 1997) is an attempt to parameterize components in order to reuse them in different contexts. In other words, on the design for variability, components must be tuned to different application requirements driven by the target applications, which makes the variability of the system a crosscutting concern.

One strategy to reduce dependencies in the use of a common set of components or data. This approach, hwoever has limitations. According to (Stevens, Myers et al. 1999) *"Every element in a common environment, whether used by particular modules or not, constitutes a separate path along which errors and changes can propagate. Each element in the common environment adds to the complexity of the total system to be comprehended by an amount representing all positive pairs of modules sharing that environment. Changes to, and new uses of, the common area potentially impact all modules in unpredictable ways. Data references may become unplanned, uncontrolled, and even unknown"*. A pub/sub system is an example of such common environment. More particularly, the events being routed, represent this common environment.

2) Proof from analysis of existing systems.
Show that according to this characterization, pub/sub infrastructures have a list of implicit or explicit inter-dependencies between those design dimensions, that are essential to the problem. For example, the event model is basic, it is fundamental and influences all the other dimensions. It is not a crosscutting concern (what is the definition of crosscutting concern in AOP?), but it is a basic assumption that impacts all other dimensions of the system.
In order to show it, I need to run software dependency metrics on existing pub/sub infrastructure source code. In this process, I can use Ariadne to understand the co-evolution of those dimensions in a real software as Siena. For example, which dimensions change when a new feature is added to the system? This feature can be the ability to provide partial ordering. Run the same thing on other open source tools.

3) Generalization and decision support.
After studying existing pub/sub infrastructures and validating and refining the model, I will be able to answer the question: given a certain problem (pubs/sub infrastructure that I want to build), show which strategy do adopt, whether traditional OO, AOP, Components or other approaches, based on the cost (lines of code, analysis and so on), to build the required system.

Indirectly, this may mean that I will be able to pick one or another versatility strategy based on how well they address my modularization criteria, or how much work is required from my generalized solution to obtain the proposed infrastructure, with a set of features required.

4) Validation

In parallel with that theoretical contribution, we need to implement a prototype, a solution that addresses the problem of co-evolution of the pub/sub infrastructure dimensions. Aspect-Oriented programming seems to be a good choice to the problem since the inter-dependencies between those modules and their concerns can be modularized by the adoption of such approach. Based on that we propose:

i) The addressing of this problem by means of aspects (Aspect-Oriented architecture)
ii) The implementation of at least three different pub/sub infrastructures that are functionally-equivalent to existing pub/sub systems (pick 3 from different domains)
iii) Show by coupling and cohesion metrics and NOV metric, that the proposed AOP solution yields in more modular solutions than the existing implementations.

DETAILS (Before talking to Crista)

What makes the development of a versatile pub/sub system a hard problem?

It is the co-evolution of all the variability dimensions. Rosemblum and Wolf (Rosenblum and Wolf 1997) divides the design dimensions of a pub/sub system into:

1. an *object model*, which characterizes the components that generate events and the components that receive notifications about events;

2. an *event model*, which provides a precise characterization of the phenomenon of an event;

3. a *naming model*, which defines how components refer to other components and the events generated by other components, for the purpose of expressing interest in event notifications;

4. an *observation model*, which defines the mechanisms by which event occurrences are observed and related;

5. a *time model*, which concerns the temporal and causal relationships between events and notifications;

6. a *notification model*, which defines the mechanisms that components use to express interest in and receive notifications; and

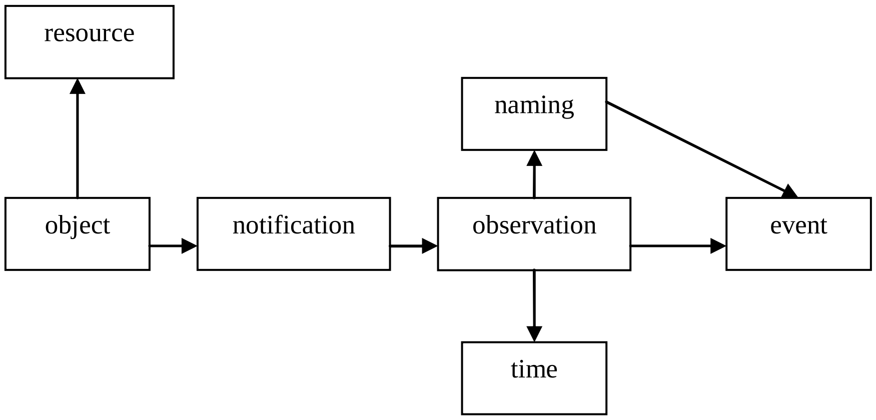
7. a *resource model*, which defines where in the Internet the observation and notification computations are located, and how resources for the computations are allocated and accounted.

Those dimensions can be used to represent the variability of different pub/sub infrastructures. The building of a generalized solution that allows the co-evolution of those dimensions is hard due to the interdependencies that exist between those dimensions. For example, the observation and notification models are entirely dependent on the event model. If one wants to modularize those first two dimensions, they must do so making assumptions about the way the events are represented and the guarantees the system provides in terms of timing. A change in one of those dimensions may invalidate all the componentization effort done in the first two dimensions (observation and notification). For example, the observation and notification model can be modularized in components that perform specific event filtering. In fact, they can be composed in event correlation hierarchies as shown by (Pietzuch, Shand et al. 2004). However, they may not be valid if the timing or the event models are changed, requiring the possible rewriting of those components to be able to handle different representations.

In an analogy with building construction, the dependent dimensions represent the base and the pillars that support the building. While division walls may be configured in different ways, the supporting pillars and the foundation cannot be changed without deeply impacting the construction of the building.

Other dimensions such as the resource and the object models may vary without significantly affecting the way events are observed or represented. Hence, they can be easier modularized or implemented with the help of different technologies such as CORBA, RMI, and different programming languages.

The following figure presents the Rosemblum and Wolf's dimensions and their conceptual inter-dependencies.



Dimensions that are inter-dependent from one another are harder to be modularized since they need to cope with all the possible variabilities in other dimensions. For example, in order to be portable and reusable in different contexts, event filtering strategies need to account for different event representations. A content-based filter, for instance, need to support different event types (attribute/value pairs, records, plain text files and so on). In other words, the variability in one dimension is directly proportional to its inter-dependency in other dimension.

Consider *d* as the number of dimensions one variability dimension *v* depends upon. In order to support the co-evolution of those dimensions, *v* needs to account for those variability dimensions. In the worst case, the complexity of implementing *v* will be proportional to all possible variations in each one of the *d* dimensions it depends upon. This relation is expressed by the formula:

$$\boxed{Complexity \sim v*var(d)}$$

Hence, in order to design a system for variability, one should try to minimize the coupling among those dimensions. Depending on the domain, however, the complete decoupling of those dimensions is not possible. As a consequence, one cannot modularize dimensions in such a way that those modules are independent from the other dimensions.

This problem has been observed throughout the computing and organizational history as the modularization problem. Parnas tries to address this problem by proposing the concept of information hiding in his seminal paper. It is a strategy to achieve the decomposability of software systems into parts that can be more easily replaced and interchanged. In order word, the breaking of code into manageable pieces that can be produced and evolved separately. The ultimate goal, however, is the minimization of the interdependencies between those modules that compose the main system. Modularization itself, does not guarantee that the internals of those modules depend on other modules.

"In a word of change, modularity is generally worth the costs. The real issue is normally not whether to be modular but how to be modular. Which modularization, which structure of encapsulation boundaries, will

yield the best system decomposition? The goal is clearly to find the modularization that minimized interdependencies and most cleanly decomposes the system." (Langlois 2000)

References

Eder, J., G. Kappel, et al. (1992). Coupling and Cohesion in Object-Oriented Systems. Information and Knowledge Management, Baltimore, USA.

Langlois, R. N. (2000). Modularity in Technology and Organizations. Research Paper 1/00. Storrs Mansfield, The University of Connecticut**:** 1-49.

Maeda, C., A. Lee, et al. (1997). Open implementation analysis and design. 1997 symposium on Software reusability, Boston, MA, ACM Press.

Parnas, D. L. (1972). On the Criteria to Be Used in Decomposing Systems into Modules. Communications of the ACM. **15:** 1053-1058.

Pietzuch, P. R., B. Shand, et al. (2004). Composite event detection as a generic middleware extension. IEEE Network Magazine. Special Issue on Middleware Technologies for Future Communication Networks. **18:** 44-55.

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

Stevens, W. P., G. J. Myers, et al. (1999). "Structured design." IBM Systems Journal **38**(2-3): 231 - 256.