A Green Family: Generating Publish/Subscribe Middleware Configurations

Bencomo N., Sivaharan T., and Blair G
Lancaster University, Computing Department., InfoLab21,
Lancaster, UK, LA1 4WA

{nelly, t.sivaharan, gordon}@comp.lancs.ac.uk

ABSTRACT

We are investigating how to combine modelling, meta-modelling, and reflection to systematically generate middleware configurations that can be targeted at different application domains and deployment environments. We have developed a set of meta-models that captures the fundamental concepts and forms the basis for systematic generation of extensible "middleware families" that can be instantiated differently depending on different requirements. GREEN [[5]] is one of the families of middleware developed at Lancaster University. GREEN is a configurable and reconfigurable publish-subscribe middleware to support pervasive computing applications. In this article we investigate, how we can systematically generate valid GREEN configurations based upon requirements and constraint specifications; thus eliminating the heavy burden on the application programmer while guaranteeing that the ultimately configured middleware will offer the required functionality.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Software Architecture; D.2.11 [**Software Engineering**]: Design; D.4.7 [**Operating Systems**]: Organization and Design – *Distributed Systems, Hierarchical design.*

General Terms: Design

Keywords: Family Generation, Model-Driven Development, Meta-models, Publish/Subscribe, Reflective Architectures.

1. INTRODUCTION

Adaptability is one important criterion for middleware platforms. In order for middleware platforms to be adaptive, their services and properties need to support a wide variety of application domains, deployment environments, and QoS properties. At Lancaster University, our middleware research group has been investigating how to combine the notions of components [9] (language-independent units of dynamic deployment), component frameworks CF (collections of components that address a specific area of concern and accept additional plug-in components) [9], and middleware families (collections of component frameworks that are tailored to specific application domains and deployment environments). We apply our approach [1][2] to generate both

deployment time and runtime re-configuration. Reflection [7] is applied to discover the current structure and behaviour of the component configurations, and to allow selected changes at run-time for dynamic adaptation. The end result is a flexible middleware platform that can be straightforwardly specialized to a wide range of domains including multimedia, embedded systems [3], and mobile computing [4].

New challenges have emerged from the experience of working with highly configurable middleware platforms. Being responsible for middleware composition and integration adds considerable complexity to the load on the application developer. Application developers have to deal with a large number of complex variability decisions when planning middleware configurations to suit the requirements. These include decisions such as what kinds of components are required and how these components must be configured together. The high flexibility offered increasingly makes it error-prone when application programmers manually create the required configurations. Such ad hoc approaches do not offer formal foundations for verification that the ultimately configured middleware will offer the required functionality.

We advocate the use of Model-Driven Software Development (MDSD) techniques to overcome the issues mentioned above. MDSD is a new paradigm that encompasses domain analysis, meta-modelling and model-driven code generation. We are shifting from the development of single middleware systems to family of middleware systems based on the meta-models and models designed. We have already specified a kernel UML meta-model that embraces the set of fundamental concepts of the component model. All middleware family members regardless of their domain shares this minimum set of concepts. On top of this, we propose a set of extensions (caplets and reflective extensions) that captures the extensibility of the approach [2]. In this article we investigate how our approach can be used in the specification and automatic generation of GREEN family configurations [5]. GREEN is a highly configurable and reconfigurable publish-subscribe middleware platform developed at Lancaster to support pervasive computing applications.

2. THE GREEN FAMILY: a Publish-Subscribe Middleware Platform

GREEN (Generic & Re-configurable EvEnt Notification service) is a highly configurable and reconfigurable publish-subscribe middleware [5] to support pervasive computing applications. Such applications must embrace both heterogeneous networks and heterogeneous devices: from embedded devices in wireless ad-hoc networks to high-power computers in the Internet. GREEN offers a flexible middleware which is highly configurable and re-configurable to meet the application requirements and constrains of the heterogeneous and changing environments. GREEN

follows the well established approach to the development of reflective middleware [10] it uses the marriage of OpenCOM components [7], reflection [8] and component frameworks (CFs) to yield a configurable, reconfigurable and evolvable publish-subscribe middleware architecture. In particular, GREEN is configurable to operate over heterogeneous network types (e.g. MANET and WAN) and supports pluggable publish-subscribe interaction types (i.e. topic based, content based, context, composite events). Further, the underlying event routing mechanisms are reconfigurable to support selected interaction type for different network types. In addition, the distributed event routing and event filtering is underpinned by pluggable distributed event broker overlays; we create overlays of event brokers to suit contrasting network types. The high configurability and flexibility of GREEN gives a good example of the new challenges that arise when planning the appropriate middleware. Developers face the responsibility of middleware composition and configuration to meet the specific application and environment requirements, which in itself add considerable complexity. The system integrity can be easily compromised if application developers fail to configure the middleware without expertise and understanding of the middleware itself.

3. THE APPROACH: A SUMMARY

In outline, different configurations are generated from the models and meta-models specified. The particular domain and the specific required functionality will shape the primary configuration. The ultimate concrete configurations are determined by three dimensions of variability we have identified:

(i) deployment environment, (ii) QoS, and (iii) degree of (re)configurability.

As the context is about the P/S paradigm, the configurations of components are related to modelling elements associated with the GREEN architecture. This architecture comprises the interaction CF and the Event Broker Overlays CF and their different plug-ins, see Figure 1. Variability associated with the deployment environment dimension is categorised in terms of network types (e.g. MANET, WAN) and device types (e.g. PDA, PC). According the P/S interaction type required, the Interaction CF is configured by plugging different interaction types (topic-based, content-based, and context-based). Finally, the Overlays CF are configured to provide different overlays plug-ins depending on the interaction and network type determined before. The overlay plug-in configured for a particular interaction type heavily influences 1) the suitability of the interaction type to the network environment, 2) scalability, and 3) fault tolerance properties of the system. Consequently, the configurations of the plug-ins of both, Interaction and Overlays CF described above, have to be done in light of configurations requirements and general QoS like fault tolerance and throughput. These conditions are taken into account when gluing together the appropriate components to obtain the optimal configurations that meet the requirements.

The partial results shown in this article make us think that we are on the right path to reach our goal. However, further work has to be done in terms of the implementation and formal evaluation of the systematic generation of GREEN family members we have proposed.
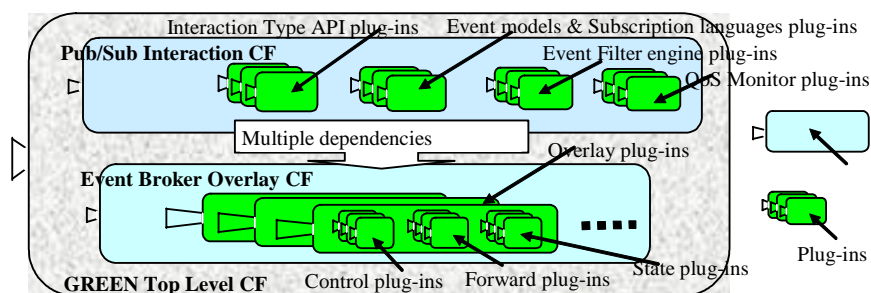

Figure 1. The GREEN Architecture

4. REFERENCES

[1] Bencomo N., and Gordon B. *Preening: Reflection of Models in the Mirror,* The Doctoral Symposium, MODELS 2005, Jamaica, October, 2005

[2] Bencomo, N., Blair, G.S., Coulson, G., Batista, T., *Towards a Meta-Modelling Approach to Configurable Middleware*, Proc. 2nd Workshop on Reflection, AOP and Meta-Data for Software Evolution, at ECOOP, 2005

[3] Costa, P., Coulson, G., Mascolo, C., Picco, G.P., Zachariadis, S.: *The RUNES Middleware: A Reconfigurable Component-based Approach to Networked Embedded Systems*, PIMRC05,(2005)

[4] Grace P., Blair G. Samuel S.: *ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability*, Proc in International Symposium on Distributed Objects and Applications (2003)

[5] Sivaharan, T., Blair, G.S., Coulson, G., *GREEN: A Configurable and Re-Configurable Publish-Subscribe Middleware for Pervasive Computing*, to appear Proc. DOA'05, Cyprus, 2005.

[6] Smith B.: *Reflection and Semantics in a Procedural Language,* PhD thesis, MIT Laboratory of Computer Science, 1982

[7] Coulson, G., Blair, G.S., Grace, P., Joolia, A., Lee, K., Ueyama, J., *OpenCOM v2: A Component Model for Building Systems Software*, Proceedings of IASTED, SEA'04, 2004

[8] Kon, F., Costa, F., Blair, G.S., Campbell, R., *The Case for Reflective Middleware: Building Middleware that is Flexible, Reconfigurable, and yet simple to Use,* CACM, Vol. 45, No. 6, pp 33-38, 2002.

[9] Szyperski, C.,*Component Software: Beyond Object-Oriented Programming.* Addison Wesley, 1998

[10] Reflective Middleware Group at Lancaster:
http://www.comp.lancs.ac.uk/computing/research/mpg/reflection/index.php