Models, Reflective Mechanisms and Family-based Systems to Support Dynamic Configuration

Bencomo N., Blair G., and Grace P.
Lancaster University
Comp. Dep., InfoLab21,
Lancaster, UK, LA1 4WA
{nelly, gordon, gracep}@comp.lancs.ac.uk

ABSTRACT
Middleware platforms must satisfy an increasingly broad and variable set of requirements arising from the needs of both applications and underlying systems deployed in dynamically changing environments such as environment monitoring and disaster management. To meet these requirements, middleware platforms must offer a high degree of configurability at deployment time and runtime. At Lancaster we use reflection, components and component frameworks, and middleware families as the basis of our approach to develop dynamically configurable middleware platforms. In our approach, components and component frameworks provide structure, and reflection provides support for dynamic configuration and extensibility for run-time evolution and adaptation. This approach however has contributed to make the development and operation of middleware platforms even more complex. Middleware developers deal with a large number of variability decisions when planning (re)configurations and adaptations. This paper examines how Model-Driven Engineering (MDE), Domain Specific Languages (DSLs) and System Family Engineering can be used to improve the development of middleware families, systematically generating middleware configurations from high level descriptions. We present Genie, a DSL-based prototype development-tool that supports the specification, validation and generation of artefacts for component-based reflective middleware. In particular, this paper describes how the Genie toolkit improves the development of the Gridkit middleware through the modelling and automated generation of middleware policies; that remove the complexity of handling large number of runtime adaptation policies.

Categories and Subject Descriptors
D.2.10 [**Software Engineering**]: Design; D.2.13 [**Software Engineering**]: Reusable Software - *Reuse models*; D.4.7 [**Operating Systems**]: Organization and Design – *Distributed Systems.*

General Terms: Design

Keywords: Reflective Middleware, Grid Computing, MDE, DSL, System Families

1. INTRODUCTION
Middleware platforms must satisfy an increasingly broad and variable set of requirements arising from the needs of both applications and underlying systems deployed in dynamically changing environments such as environment monitoring and disaster management. Even more, such platforms must embrace both heterogeneous networks and heterogeneous devices: from embedded devices in wireless ad-hoc networks to high-power computers in the Internet. To meet these requirements, middleware platforms must offer a high degree of configurability at deployment time and at runtime time. Significant advances have been made in recent years in the general area of dynamic reconfiguration [1, 12, 22, 24, 26, 27, 31].

At Lancaster we use reflection, components and component frameworks, and middleware families as the basis of our approach to develop dynamically configurable middleware platforms. We have successfully developed several experimental reflective middleware platforms and applications [15-17, 21]. In our approach, components and component frameworks provide extendable structure and functionality, and reflection offers the essential support for dynamic configuration and extensibility for run-time evolution and adaptation. Currently, middleware developers deal with a large number of variability decisions when planning (re)configurations and adaptations. This approach however has contributed to make the development and operation of middleware platforms even more complex. Middleware developers deal with a large number of variability decisions when planning (re)configurations and adaptations. This large number makes it error prone to manually guarantee that all the decisions are consistent. Such ad hoc approaches do not offer formal foundations for verification that the middleware will offer the required functionality [5, 6]. We strongly believe that dynamically reconfigurable middleware platforms require new software development and operational paradigms to support systematic and automated checking of both functional and non functional properties.

In this paper, we describe our experience of how Model-Driven Engineering (MDE), and specifically Domain Specific Languages (DSLs), and System Family Engineering can be used to improve the development of middleware families, systematically generating middleware configurations from high level descriptions. To demonstrate our approach we describe how the Gridkit middleware[10, 15] development is improved by the Genie tool [3]. Genie is a prototype development-tool platform that offers a Domain Specific Language for the specification,

validation and generation of artefacts for component-based reflective middleware. Gridkit is an experimental reflective middleware for grid computing, which supports adaptive grid and pervasive applications by dynamically combining middleware behaviour (composed as component frameworks) based on application requirements and changing environmental context. Specifically, we illustrate how the Genie toolkit improves the development of the Gridkit middleware through the modelling and automated generation of middleware policies; that remove the complexity of handling large number of adaptive policies.

The paper is organized as follows. In Section 2 the Gridkit middleware is introduced. Section 3 discusses how Model Driven Engineering can be applied to improve the development of middleware families by systematically and automatically generating the middleware families. Section 4 introduces Genie and in particular, describes how Genie can be used to support configurability and re-configurability of one particular adaptive (and reflective) technology, i.e. the Gridkit. In section 5, we discuss our approach in the context of related work. Section 6 provides a summary and outlook.

2. GRIDKIT MIDDLEWARE

Gridkit is one of the dynamically configurable middleware families that have been developed using the OpenCOM component model [8]. A key feature of Gridkit is its ability to handle diversity i.e. it can be dynamically configured to meet different application requirements in different environmental settings; hence, different middleware services are tailored to the current domain. For example, Gridkit can offer a publish-subscribe service in an ad-hoc network, or it can be an Object Request Broker in the fixed network. This high-level of variability makes it an ideal candidate for model driven development techniques.

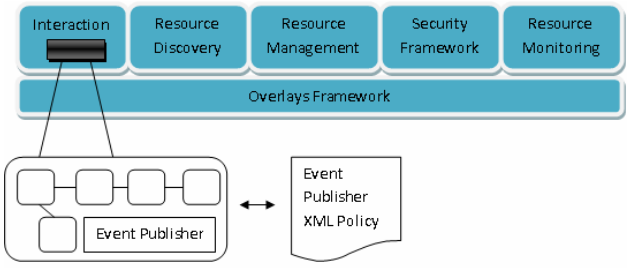

Figure 1: The Gridkit Middleware

Fundamentally, the Gridkit architecture is composed of component frameworks, where a component framework is a management unit for a set of components related to a particular domain of middleware behaviour. Like components these can be composed and connected to build a suitable middleware for the particular set of requirements. Figure 1 illustrates the key Gridkit frameworks. The overlays framework supports multiple virtual network services (e.g. multicast, ad-hoc routing) required by higher level frameworks. Above the overlays framework is a set of further "vertical" frameworks that provide functionality in various orthogonal areas, and can optionally be included or not included on. In brief, the frameworks are as follows: the *interaction framework* accepts multiple interaction type plug-ins (e.g. RPC, publish-subscribe, group communication); the *resource discovery framework* accepts plug-in strategies to discover application services (e.g. SLP, UPnP, Salutation) and resources such as CPUs and storage (e.g. peer-to-peer search); the *resource management and resource monitoring frameworks* are respectively responsible for managing and monitoring resources; and the *security framework* provides general security services for the rest of the frameworks. These frameworks are discussed in more detail in [11].

Figure 1 also illustrates how middleware configurations are chosen and deployed dynamically at runtime. Frameworks are also the unit of adaptation; policies for configuration and reconfiguration are related to individual frameworks; that is, an instance of the interaction framework has one set of policies, while the overlay framework will use a different set of policies. We advocate this approach to allow domain specific policies to be included by different developers. These policies are declarative XML statements written by middleware developers. The configuration policies describe framework configurations for meeting a given set of requirements in a given set of environmental conditions. Typically, these policies are in the format: *for requirement set S, on condition set C, configure component graph A.* In figure 1 the interaction framework is specialized to an event publisher personality described by the chosen policy.

To illustrate the dynamic configurations of Gridkit, consider the roaming of a fire fighter [9] from a part of the forest covered by an infrastructure network, to a part served only by an ad-hoc network. When the application is deployed in the infrastructure network, policies for the interaction framework and overlay framework tailor an event broker optimized for wired behaviour, whereas when the application is deployed in the ad-hoc environment the policies define a different event broker more suited to ad-hoc routing behaviour.

3. MODEL_DRIVEN ENGINEERING TO GENERATE MIDDLEWARE FAMILIES

Gridkit reflects the modern view of middleware [11, 31], that a set of middleware capabilities needs to be tailored to classes of problem domains that are increasingly demanding advanced functionality such as the ability to adapt dynamically. The disadvantage of this approach is the cost of producing different middleware systems. The OpenCOM component model supports this task but still entails substantial work to instantiate and configure a given systems architecture. For example, the Publisher framework, only includes 4 components and 5 bindings; however the associated XML file that describes the configuration has 64 lines. The lines of these kinds of files grow exponentially depending on the number of components and bindings. This also ignores the extra code required to manage change at run-time. Therefore, an important strand of our current research is to investigate how generating instances of middleware tailored to specific problem domains and contexts can be achieved in a more systematic, consistent and, if possible, automatic way.

We propose a model-driven engineering (MDE) approach [23] to help overcome this problem. The first step in this process involves the use of UML to specify a set of meta-models representing the core middleware functionality and the reflective

functionality that is common to all middleware family members regardless of their domain[5, 6]. In effect, the domain addressed by these meta-models is the more generic one of reflective, adaptive middleware and represents the fundamental component-based concepts that underpin our architecture. The syntax and semantics offered by UML are sufficient to model the OpenCOM concepts (more details of these meta-models can be found in [6]). However, when specifying concepts related to higher level abstractions related to domains of application, or in our case particular domains within the middleware itself (e.g. the interaction domain), more specific modeling concepts are needed. For example, a modeling language for developing and assembling OpenCOM Grid oriented platforms should contain concepts like overlay network frameworks, resource management frameworks, and resource discovery frameworks; concepts clearly required to describe valid configurations of the platform. Therefore we advocate the development of a range of DSLs for different application domains: OpenCOM itself, Grid computing, Publish/Subscribe, Mobile Computing, Multimedia, etc. The DSLs map directly on to underlying OpenCOM concepts (components, component frameworks, etc). They are then used to automate or semi-automate generation of artefacts (source code, configuration descriptors, etc) related to the development, deployment, and configuration of the different middleware platforms.

The need of modelling run-time information

In the MDE area, research has focused mainly on using models at design, implementation, and deployment stages of development. These efforts have been prolific with many tools and technologies already succeeding in the industry [7]. However, the ability of design-time architecture models to represent run-time information still needs research. We think that the use of model-driven techniques for validating and monitoring run-time behaviour can also engender substantial benefits. We particularly advocate the combining of models at run-time with generative techniques to produce instances of adaptive middleware (and in particular reflective middleware). A noteworthy aspect is that these models can accompany the software system and provide the basis for defining and executing run-time monitoring and reconfiguration. In the specific case of Gridkit, run-time configurations of components associated with the different components frameworks and its policies can be designed and validated in advance and off-line. This allows middleware developers to reason, plan, and validate the configurations early in the design phase and catch design faults before operation time. At run-time, Gridkit reflective mechanisms read and interpret policies to dynamically create and manage the respective valid configurations. Our approach to support the use of configurable run-time models is presented in the next section.

4. GENIE: A TOOL FOR GENERATION OF MIDDLEWARE FAMILIES

Genie is a prototype for a development-environment that offers a DSL for the specification, validation, and generation of artifacts for OpenCOM-based middleware platforms The tool has been developed using MetaEdit+ [28]. Genie allows the creation and validation of models that drive the life cycle of the reflective middleware families at Lancaster University. From the models specified not only source code can be generated but configuration files, results associated with model checking and validations, testing code and documentation.

As in other program family techniques, our approach uses component frameworks to manage and accomplish variability and development of systems that can be adapted by re-configuration. A component framework enforces architectural principles (constraints) on the components it supports; this is especially important in reflective architectures that dynamically change, and whose changes must be verified. Models associated to component frameworks are used to represent the possible versions and variants of the different families. “Configuring a system is the process of choosing a specific family instance and modifying the run-time structure of a system to conform to the chosen instance.” [20] In our approach this implies the insertion, deletion and modification of the structural elements represented by the component frameworks: components, interfaces, receptacles, binding components and constraints.

4.1 OpenCOM DSL

Existing models of OpenCOM based middleware families use a wide variety of notations that depend on the abstraction of the domain that is being modelled. However, the basic concepts of any model specified in Genie relates to components, interfaces and component frameworks as the component model OpenCOM dictates. Genie offers a common modeling notation for all the models that is called the OpenCOM DSL. The specification of how these concepts work together is described in the graphs (configurations) associated with the components and component frameworks modelled. In those models, a component offers and requires interfaces, interfaces can bind together to connect components (i.e required interfaces or receptacles in components are bond with offered interfaces of other components). A component framework is able to export interfaces from internal components. In the same way component frameworks can require interfaces to satisfy the requirements of some of their components. A partial view of a model of a component framework is shown in Figure 4. All the concepts and relations described above and contained in the OpenCOM DSL are dictated by the OpenCOM as described in [8].All middleware family members regardless of their domain share this minimum set of concepts.

4.2 Model-Driven Gridkit Policies

The pervasive computing scenarios managed by Gridkit do not remain constant and invariable over time. Therefore, to configure and maintain the same configuration is inadequate, rather the middleware must be dynamically reconfigured to cope with changes [16]. In section 2 we described the role of the middleware developer, who must write a series of configuration policies to tailor, and potentially extend the behaviour of the reflective middleware. Here, we demonstrate how these configurations are designed beforehand using Genie. Genie provides improved support for the design and validation of Gridkit policies, which can be created in a more efficient manner.

Figure 2 shows how Genie relates to Gridkit. Modellers create and design components and component frameworks to generate different artefacts including the policies in the form of configuration of components. Components are stored in the Component Repository and the policies are stored in the Knowledge Repository. At run-time Gridkit then reads the

validated policies to configure appropriate middleware personalities. To do this Gridkit downloads components from the Repository. Notice that the Gridkit framework is deployed on each participating node (a mobile device, a laptop or even a tiny system proper of embedded network systems). The repository can be centralized or distributed according the implementation.

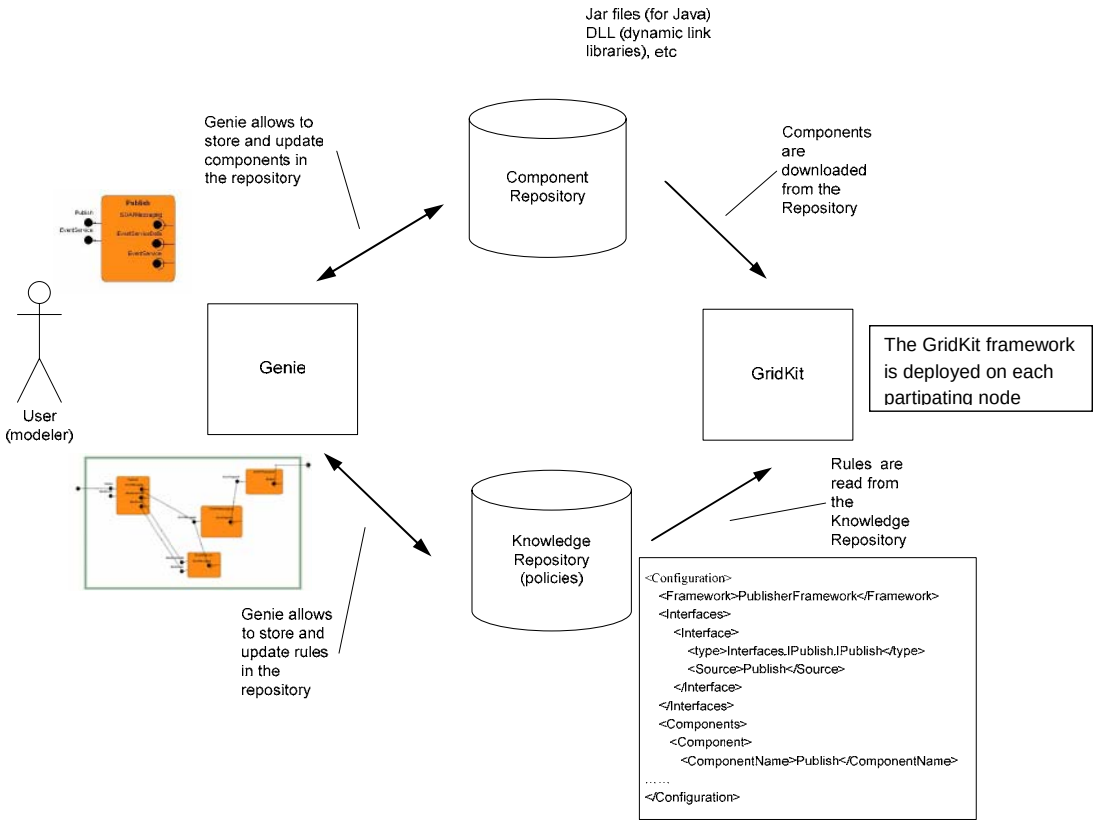

Figure 2: Genie and Gridkit

Configuration policies are scoped to individual frameworks so an instance of the interaction framework will have one set of policies, while an instance of the overlay framework will use a different set of policies; see Figure 3. These policies define how Gridkit satisfies a requirement in a given environmental context, and also how Gridkit adapts to changed environmental context.

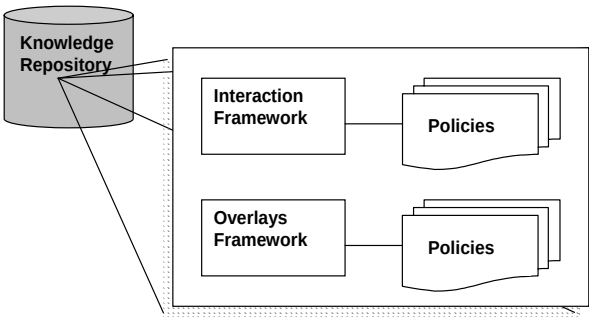

Figure 3: Gridkit Framework and Policies

Figure 4, shows a policy (XML file) generated from the component framework shown in the graph. From component frameworks models not just policies can be generated, a broad range of artefacts can be generated (components source code, test code, documentation, etc.), for more details see [4]

Any generation of artefacts, including policies do require a validation of the content in the model; this topic is covered in the next section.

4.3 Validation of Models

Constraints are intrinsic parts of the models specified in Genie and are the basis of validations and checkings. As noted above, a component framework imposes constraints on the components it supports. Consequently the fundamental checking is related to these architectural constraints. When designing the validations of the component frameworks we exploit known variabilities in architectural structures so that common checking infrastructure can be built once and then used by any user of Genie in the corresponding component framework. Not only does this approach decrease the cost of models validation, but it makes the technology easier, since the modeller needs just to be concerned about the domain-specific aspects of the problem; in this case the behaviour of components and specific domain-related constraints (architectural styles and new constraints).

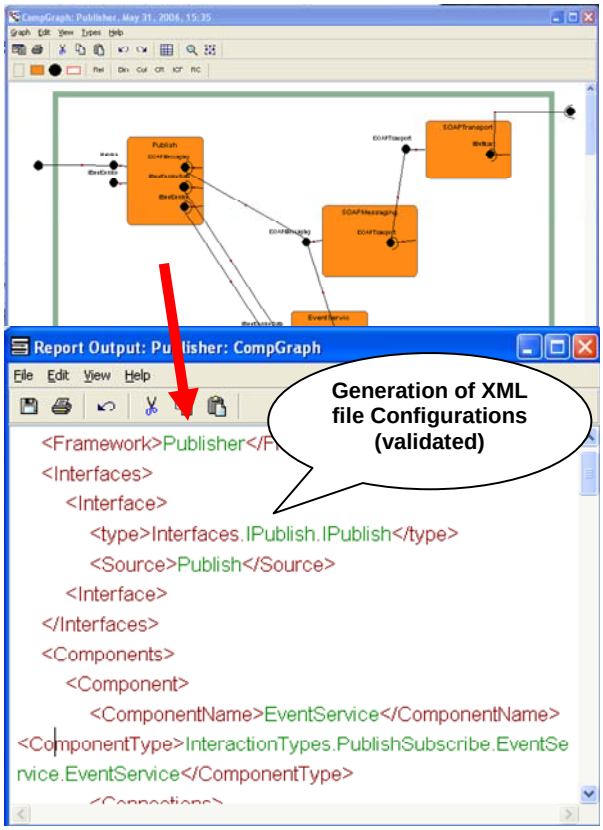

Figure 4: Generation of Policies in Genie

An example of basic validation is the verification that all the connections between required interfaces and offered interfaces conform to the same type (therefore the configurator does not need to check these conditions at run-time). Examples of more specific validations are related to the specific constraints enforced by the component frameworks: a specific component may appear only once at the most, a connection between two components must exist, etc. These validations should be written for all the component-framework models.

5. RELATED WORK

A significant number of experimental middleware platforms have been developed including Dynamic TAO, LegORB and UIC [29] (all University of Illinois at Urbana-Champaign), Open CORBA [25] (Ecole des Mines de Nantes), Flexinet [19] (APM, Cambridge), and OOPP [2] (University of Tromsø). They mainly address issues of configurable middleware and application at deployment time and have offered some valuable initiatives towards dynamic reconfigurations. However, we are unaware of any other current research for dynamic reconfiguration with the high degree of runtime reconfigurability offered by Gridkit.

An increasing number of model driven tools have been developed for modeling component-based systems. Cadena [18] an environment for building and modeling CORBA Component systems for component-based distributed real-time embedded systems. CoSMIC [14] offers domain-specific tools for composing and deploying distributed real time middleware-based applications. However these works focus on configuration at design and deployment time and do not tackle directly the problem of models at run-time for valid reconfiguration and adaptation

The MADAM project [12] aims to facilitate adaptive application development for mobile computing following an architecture-centric approach where they represent architecture models at run-time allowing components to reason about adaptation. The MADAM project as our approach moves in the direction of allowing policy developers to use reflection to specify reconfigurations. We certainly think that this way we will be able to tackle much more complex systems. MADAM, unlike our approach, focuses on tackling only mobile environments requirements; our research also focuses on other domains

J3 Toolsuite [32] an MDE tool that visually captures the design of EJB applications, their quality of service (QoS) requirements, and the autonomic properties applied to their EJBs. J3 can generate code to plug EJBs into a Java component framework that provides autonomic capabilities. J3 and our approach share the support design of high level specifications and their validations at design time. However, J3 focuses on the development of EJB applications and relies on EJB reflective capabilities. Our approach is language independent, which makes it more generic

There are plenty of UML based tools that provide modeling capabilities for component-based systems using primarily code-level abstractions. Our approach is different by also taking advantage of the abstractions in high-level design models that naturally tailor specific domains. However we think DSL and UML approaches are complementary more than contrary.

6. SUMMARY AND FUTURE WORK

In this paper we have presented our approach to generating run-time adaptation policies using Model Driven Engineering and Domain Specific Languages. These policies allow middleware developers to use reflection to specify configurations in a systematic way. Specifically, we have described the DSM-based Genie toolkit and its role in the automated generation of configuration policies for the GridKit middleware. With our approach, middleware developers reason, plan, and validate the policy-based configurations early in the design phase. Using advance validation, design faults of configurations can be caught before runtime.

Substantial research remains to be done. One of the big challenges is how to deal with the problem of the combinatorial explosion related to the number of policy-based configurations. We are also enhancing our prototype to increase its expressive and generative capabilities to include new frameworks; for example we are working on the specification of models for families of Service Discovery Protocols with a common architecture [13].

Of particular interest is the study of how our approach can be enhanced to maintain the integrity of the state of the system when performing reconfigurations at runtime. To do this we are currently investigating the formal specification and generation of policies that guide the reconfigurations managed by the middleware. Finally, we are investigating also how requirements

for domains of application can directly influence the generation of domain-specific middleware through the DSLs and meta-models [30].

Acknowledgments: Grateful acknowledgment is made to MetaCase for permission and support when using MetaEdit+.

7. REFERENCES

1. Aksit, M. and Choukair, Z., Dynamic, Adaptive, and Reconfigurable Systems Overview and Prospective Vision. in *23rd Int'l Conf. Distributed Computing Systems Workshops (ICDCSW)*, (2003).

2. Andersen, A., Eliassen, F. and Blair, G., A Reflective Component-Based Middleware with Quality of Service Management. in *Protocols for Multimedia Systems PROMS'2000*, (Poland, 2000).

3. Bencomo, N. Genie: a Model-Driven Engineering Tool for the Generation Adaptive Middleware Families. PhD Research: http://www.comp.lancs.ac.uk/computing/users/bencomo/Genie, Lancaster University, Lancaster, UK, 2006.

4. Bencomo, N. and Blair, G., Genie: a Domain-Specific Modeling Tool for the Generation of Adaptive and Reflective Middleware Families. in *Submitted to 6th OOPSLA Workshop on Domain-Specific Modeling*, (Portland, 2006).

5. Bencomo, N. and Blair, G. Raising a Reflective Family *Models and Aspects Handling Crosscutting Concerns in MDSD*, Glasgow, Scotland, 2005.

6. Bencomo, N., Blair, G., Coulson, G. and Batista, T. Towards a MetaModelling Approach to Configurable Middleware *2nd ECOOP'2005 Workshop on Reflection, AOP and MetaData for Software Evolution RAM-SE* Glasgow, Scotland, 2005.

7. Bencomo, N., Blair, G. and France, R. Models@runt.time. Workshop in conjunction with MoDELS / UML 2006, 2006.

8. Blair, G., Coulson, G., Ueyama, J., Lee, K. and Joolia, A., OpenCOM v2: A Component Model for Building Systems Software. in *IASTED Software Engineering and Applications*, (USA, 2004).

9. Cooper, C., Duce, D., Younas, M., Li, W., Sagar, M., Blair, G., Coulson, G. and Grace, P., The Open Overlays Collaborative Workspace. in *UK E-Science All Hands Meeting*, (Nottingham, UK, 2005).

10. Coulson, G., Grace, P., Blair, G., Cai, W., Cooper, C., Duce, D., Mathy, L., Yeung, W.K., Porter, B., Sagar, M. and Li, W. A Component-based Middleware Framework for Configurable and Reconfigurable Grid Computing. *Concurrency and Computation: Practice and Experience*, *18* (8). 865-874.

11. Eliassen, F., Andersen, A., Blair, G.S., Costa, F., Coulson, G., Goebel, V., Hansen, Ø., Kristensen, T., Plagemann, T., Rafaelsen, H.O., Saikoski, K.B. and Yu, W., Next Generation Middleware: Requirements, Architecture, and Prototypes. in *7th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'99)*, (South Africa, 1999).

12. Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K. and Gjorven, E. Using Architecture Models for Runtime Adaptability. *Software IEEE*, *23* (2). 62-70.

13. Flores, C., Blair, G. and Grace, P., Service Discovery in Highly Heterogeneous Environments. in *4th Minema Workshop*, (Lisbon, Portugal, 2006).

14. Gokhale, A.S., Schmidt, D.C., Lu, T., Natarajan, B. and Wang, N., CoSMIC: An MDA Generative Tool for Distributed Real-time and Embedded Applications. in *Middleware Workshops 2003*, (Brasil, 2003).

15. Grace, P., Coulson, G., Blair, G., Mathy, L., Duce, D., Cooper, C., Yeung, W.K. and Cai, W., GRIDKIT: Pluggable Overlay Networks for Grid Computing. in *Symposium on Distributed Objects and Applications (DOA)*, (Cyprus, 2004).

16. Grace, P., Coulson, G., Blair, G. and Porter, B., "Addressing Network Heterogeneity in Pervasive Application Environments", . in *1st International Conference on Integrated Internet Ad-hoc and Sensor Networks (Intersense 2006)*, (Nice, France, 2006).

17. Greenwood, P., Hughes, D., B, B.P., Grace, P., Coulson, G., Blair, G., Taiani, F., Pappenberger, F., Smith, P. and Beven, K., Using Grid Technologies to Optimise a Wireless Sensor Network for Flood Management. in *4th ACM Conference on Embedded Networked Sensor Systems (Demo Session)*, (Colorado, USA, 2006).

18. Hatcliff, J., Deng, W., Dwyer, M., Jung, G. and Ranganath, V.P., Cadena: An Integrated Development, Analysis, and Verification Environment for Component-based Systems. in *International Conference on Software Engineering (ICSE)*, (Portland, 2003).

19. Hayton, R., Herbert, A. and Donaldson, D., FlexiNet: A Flexible Component-oriented Middleware System". in *8th ACM SIGOPS European Workshop on Support for Composing Distributed Applications*, *Sintra*, (1998).

20. Heimbigner, D. and Wolf, A. Intrusion Management Using Configurable Architecture Models. Technical Report, Department of Computer Science, University of Colorado, Colorado, USA, 2002.

21. Hughes, D., Greenwood, P., Coulson, G., Blair, G., Pappenberger, F., Smith, P. and Beven, K., GridStix:: Supporting Flood Prediction using Embedded Hardware and Next Generation Grid Middleware. in *4th International Workshop on Mobile Distributed Computing (MDC'06)*, (Niagara Falls, USA, 2006).

22. Keeney, J. Completely Unanticipated Dynamic Adaptation of Software *Department of Computer Science*, Trinity College Dublin, 2004.

23. Kent, S., Model Driven Engineering. in *Third International Conference on Integrated Formal Methods (IFM 2002)*, (Turku, Finland, 2002), Springer-Verlag, 286-298.

24. Kon, F., Costa, F., Blair, G. and Campbell, R. The case for reflective middleware. *Communications of the ACM*, *45* (6). 33-38.

25. Ledoux, T., OpenCorba: A Reflective Open Broker. in *Reflection'99*, (France, 1999).

26. Magee, J. and Kramer, J., Dynamic structure in software architectures. in *4th ACM sigsoft symposium on Foundations of Software Engineering (FSE)*, (California, 1996).

27. McKinley, P.K., Sadjadi, S.M., Kasten, E.P. and Cheng, B.H.C. Composing Adaptive Software. *IEEE Computer*, *37* (7). 56-64.

28. MetaCase. Domain-Specific Modeling with MetaEdit+ (http://www.metacase.com/).

29. Roman, M., Kon, F. and Campbell, R.H. Reflective Middleware: From the Desk to your Hand. *IEEE DS Online, Special Issue on Reflective Middleware*, *2* (2).

30. Sawyer, P., Bencomo, N., Grace, P. and Blair, G., Ubiquitous Computing: Adaptability Requirements Supported by Middleware Platforms. in *Workshop on Software Engineering Challenges for Ubiquitous Computing*, (Lancaster, UK, 2006).

31. Schmidt, D.C., Schantz, R.E., Masters, M.W., Cross, J.K., Sharp, D.C. and DiPalma, L.P. Toward Adaptive and Reflective Middleware for Network-Centric Combat Systems. *Crosstalk The Journal of Defense Software Engineering*. 10-16.

32. White, J., Schmidt, D.C. and Gokhale, A., Simplifying Autonomic Enterprise Java Bean Applications via Model-driven Development: a Case Study. in *MoDELS 06*, (Jamaica, 2005), 601-615.