Genie: Supporting the Model Driven Development of Reflective, Component-based Adaptive Systems

Nelly Bencomo, Paul Grace, Carlos Flores, Danny Hughes, Gordon Blair
Computing Department, InfoLab21, Lancaster University, LA1 4WA, United Kingdom
{nelly,gracep,floresco,danny,gordon}@comp.lancs.ac.uk

ABSTRACT

Engineering adaptive software is an increasingly complex task. Here, we demonstrate *Genie*, a tool that supports the modelling, generation, and operation of highly reconfigurable, component-based systems. We showcase how Genie is used in two case-studies: i) the development and operation of an adaptive flood warning system, and ii) a service discovery application. In this context, adaptation is enabled by the Gridkit reflective middleware platform.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design; D.2.13 [**Software Engineering**]: Reusable Software—*Domain Engineering, Reuse models*

General Terms

Design

Keywords

Model-driven Engineering, Reflective Middleware, Dynamic Variability, Software Generation

1. INTRODUCTION

It is becoming common that systems are required to dynamically reconfigure and adapt according to context fluctuations during runtime. One approach to handling this requirement is to augment systems with intrinsic component-based, reflective, and adaptive capabilities. An example of this is *reflective middleware* [8], which supports the operation of distributed systems in fluctuating environmental conditions; the middleware can be configured to operate in different domains, and adapted to react to fluctuating environmental conditions. Such middleware is typically constructed using *components* and the associated concept of *component frameworks* to provide extendable structure and functionality; while reflection underpins dynamic configuration and extensibility for runtime evolution and adaptation.

However, the added flexibility and adaptation capabilities contribute to making the development and operation of such systems increasingly complex. Developers must deal with a large number of variability decisions when planning the configurations, reconfigurations and adaptations. These include decisions such as what components are required, and how these components must be configured according to variations in the environment and context. The above leads to a situation where variability management is an important concern. Variability management means that a systematic approach to structure, implement and document the variability in a software family should be realized in a repeatable manner. Managing variability requires a scalable and consistent approach that exploits reuse independently from specific contexts, and avoids ad-hoc solutions. Furthermore, the highly technical knowledge needed by developers requires them to work at very low levels of abstraction. This creates a big gap between the way domain experts, architects and programmers operate. Ad-hoc approaches do not offer formal foundations for verification that the systems will offer the required functionality.

Hence, we have developed an approach to address the challenges described above [1, 3, 11, 2]; this systematically promotes software reuse and use of models as first class entities to raise the level of abstraction beyond coding, by specifying solutions using domain concepts. Moreover, the approach offers a structured management of variability. In this paper, we elaborate on the approach proposed and illustrate its implementation, the *Genie* tool. Genie offers domain-specific languages (DSLs) for the modelling and generation of adaptive systems supported by the reflective middleware platforms. *Genie* has been implemented using MetaEdit+ [9].

The reminder of this paper is structured as follows: in Section 2 the reflective middleware philosophy and Gridkit are introduced. Section 3 gives an overall description of the approach implemented by *Genie* and describes the tool *Genie* itself. Section 4 briefly describes the demonstration. Section 5 concludes and discusses possible future extensions.

2. BACKGROUND

At Lancaster University, we have gained experience developing adaptive systems and middleware platforms using component frameworks and reflective technologies [4]. *Component frameworks* are management units for a set of components that address a specific domain of concern and accept "plug-in" components that add or extend behaviour. Examples of domains are routing algorithms (provided by the *spanning tree framework*) and discovery services strategies (provided by the *resource discovery framework*). Like components these can be composed and connected to build a suitable systems for a particular set of requirements. Reflective capabilities support introspection to observe and reason about the state of the system to make decisions on architectural reconfigurations. Adaptive behavior is defined by sets of reconfiguration policies. These policies are of the form *on-event-do-actions* and actions are architectural changes using the component frameworks. A context engine receives relevant environmental events that are employed to identify the reconfiguration policy to be used.

During execution, the system will be dynamically reconfigured from one structural variant to another according to variations in the context or environment, see Figure 1. Component frameworks offer the medium to provide *structural variability* and the reconfiguration policy-based mechanisms set the basis for dealing with *environment or context variability*. Structural variability and environment and context variability are dimensions of *dynamic variability* (also called runtime variability) and are further explained in [2]. Genie allows the modelling and generation of software artefacts that will be used by the system at start-up time and runtime.

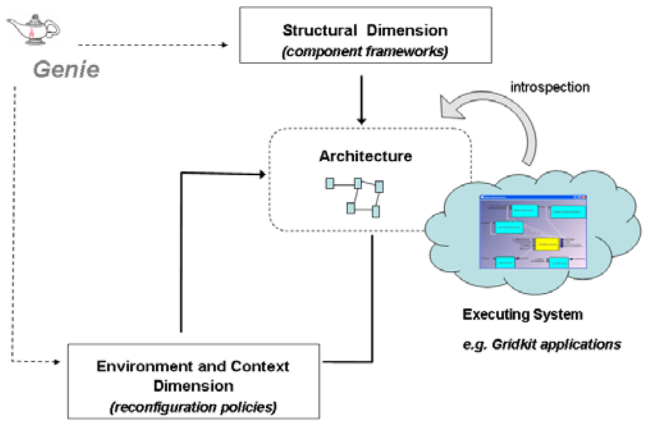

Figure 1: Dynamic Variability

Gridkit [6] is the embodiment of the reflective middleware-based philosophy briefly presented above to provide support for reconfiguration and adaptation. The case studies shown in this demo, the flood warning system GridStix[7, 11] and the service discovery application [2] are based on the Gridkit platform.

3. THE GENIE TOOL

3.1 Approach supported by Genie

The model-based approach supported by *Genie* allows developers to make use of architectural concepts using levels of abstraction beyond programming and provided by models. The three different levels of abstractions promoted by the approach are shown in Figure 2. The figure shows the artefacts that populate the layers which correspond to different levels of abstraction (abstraction levels are raised from bottom to top). Domain-specific modelling languages (DSLs for short) are used for the construction of the models associated with the structural and environment variability (at levels 2 and 3). Using these models and generative techniques, software artefacts of level 1 are generated.

Level 1 at the bottom is populated by different software artefacts like *component source code*, and *files of configurations* of component frameworks and *reconfiguration policies*. Level 2 corresponds to the *models* associated with *components* and *component frameworks* (configurations). These models provide visual representations of the component configurations. At this level, the developer/architect is able to reason about composition decisions, commonalities, and variabilities at an architectural level. At level 3 the developer reasons in terms of *structural variants* and conditions of the environment and context that trigger the reconfigurations. This level is populated by models of *transition diagrams*.

Genie models will be explained using one of the case studies of the demo, GridStix [7]. GridStix is a grid-enabled wireless sensor network for flood management that has been deployed in prototype form on the flood plain of the River Ribble in North Yorkshire, England. Level 3 of Figure 2 shows the transition diagram that guides the reconfiguration and adaptation process of GridStix. Three possible states were identified: *Normal*, *Alert*, and *Emergency*. Each state of the graph can be seen as a variant of the system and is described using two component frameworks, i.e. the *Spanning Tree* and the *Network* component framework.

The *Spanning Tree* component framework, describes the routing algorithm that has two possible variants: *Shortest Path*(SP) and *Fewest Hop*(FH). The second one, the *Overlay* component framework describes the type of network to be used and offers two possible variants: *BlueTooth*(BT) and *WiFi*. How different variants of these component frameworks are chosen will depend on the variations of conditions in the environment and context. This variation is specified using the triggers associated with the transitions in the diagram (i.e. arches). Triggers of reconfiguration policies are specified in the arches between states. The number of transitions in the transition diagrams will depend on how adaptable the system should be or is conceived.

According to the transition diagram, if the application is operating as *Normal*, and the prediction model of GridStix predicts an imminent flood (i.e. the *FloodPredicted* monitoring condition is true), the nodes adapt to the *Emergency* state bypassing the *Alert* state. This adaptation is effected by reconfiguring the *Network* to use *WiFi* instead of *BlueTooth*, and the *Spanning Tree* to a *Fewest Hop* topology.

3.2 Domain Specific Languages in Genie

Essentially, *Genie* offers two DSLs for the design of models, named the *OpenCOM DSL* [1] and the *Transition Diagrams DSL* respectively. In essence these DSLs allow the specification of the *structural variability* and the *environment or context variability* respectively:

The ***OpenCOM DSL*** allows the construction of models for components and component frameworks (configurations) that populate level 2. The modelling elements to be used are generic architectural elements such as components, required and offered interfaces, and bindings. The OpenCOM DSL can be seen as an Architecture Description Language (ADL) with generative capabilities.

The ***Transition Diagrams DSL*** allows the specification (models) of adaptations of the form: from the configuration *Ci* and on the set of conditions *Tk*, go to configuration *Cj*. These models are in essence *transition diagrams* and populate level 3. Each node in a transition diagram is considered as a *structural variant* of the system. Structural variants are "*coarser grain*" configurations than configurations associated with individual component frameworks in the sense

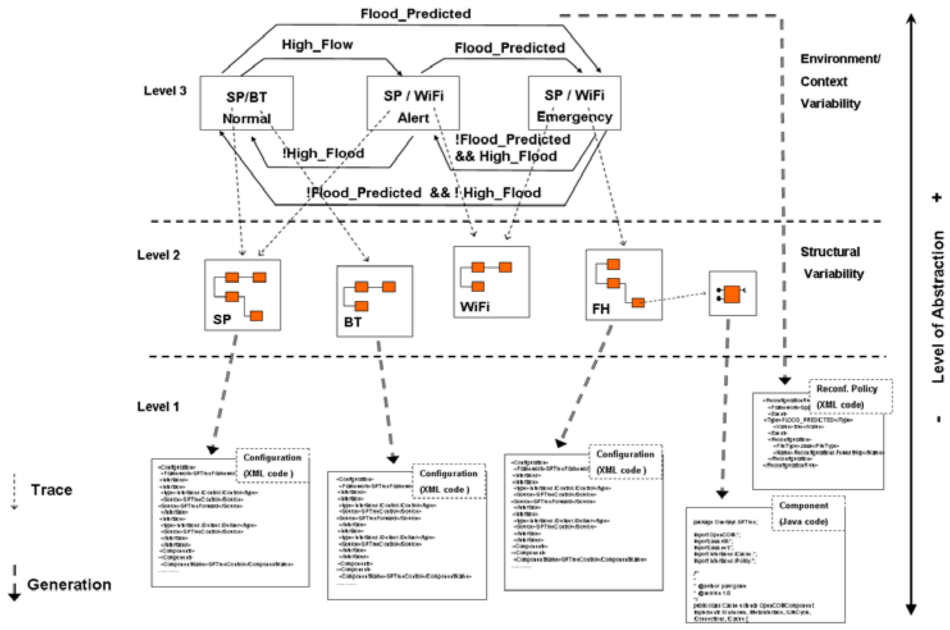

Figure 2: Overview of the approach implemented by Genie

that they are described by a set (or n-tuple) of component frameworks. Structural variants can be seen as configurations of the set of component frameworks which are associated with the problem domain. Thus, in the case study GridStix, the problem domain identified requires structural changes (in terms of reconfiguration) of the routing algorithms and the networks interfaces to be used in the sensor network. The component frameworks to be used in each structural variant should represent concepts associated with the overlays component framework, specifically the *Spanning Tree* and the *Network* framework. The 2-tuples associated with the structural variants used in the case study are (SP,BT), $(SP,WiFi)$, and $(FH,WiFi)$. The system will evolve over time according to the conditions of the environment specified in the arcs of the diagrams. The places where the architecture can be changed and the consequences of the changes will be driven by the transition diagrams.

3.3 Orthogonal Variability Models

To complement the approach described above, the orthogonal variability models proposed in [10] are used. An orthogonal variability model defines the variability of a system family in a separate model. It relates the variability specified to other software development models such as component models in our case. Figure 3 shows the variability diagrams used to model the variants in the case study. The three structural variants, *Normal*, *Alert*, and *Emergency* are associated with the variation point *VP:Flood App* marked by (a). Each state variant of the graph is described using two component frameworks, i.e. the *Spanning Tree* and the *Network* component framework as seen above. The *Spanning Tree* and the *Network* component frameworks has variation points associated themselves, marked by (b) and (c). The variability models have been particularly useful when managing the traceability relations between the structural variants of the transition diagrams (level 3) and the component frameworks configurations (level 2). This traceability relationships are fundamental for the generation of policies.

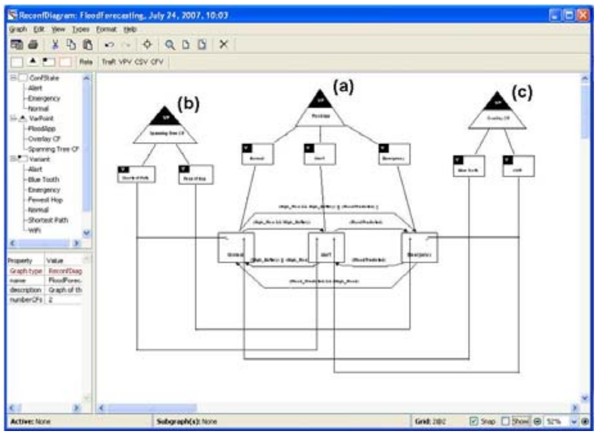

Figure 3: Variability and Transition Diagrams

3.4 Artefacts Generated by Genie

It was explained above how the developer designs models to specify the components, component frameworks and configurations, structural variants and the transition diagrams using the DSLs provided by *Genie*. Using generators that traverse these models, different software artefacts of level 3 can be generated (see Figure 4):

- From the models specified using the OpenCOM DSL components and configurations of components associated with the component frameworks are generated. To ensure consistency of the generated artefacts, the constraints specified by the models are used to validate the configurations before any generation. The middleware platforms, in this case Gridkit, allows newly generated components and component configurations to be added during the execution of the system.

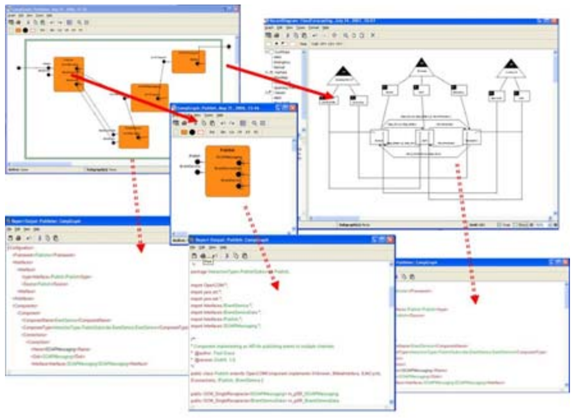

Figure 4: Genie Models and Generated Artefacts

- Using the transition models specified, the reconfiguration policies are generated. As in the case above, validation of transition diagrams should be performed to avoid inconsistencies. Gridkit allows the generated policies to be inserted during execution. The newly added reconfiguration policies are used as long as the "new" component(s) or component configuration(s) that provide the match are also provided.

4. DEMONSTRATION

Using *Genie*, we will demonstrate two specific case studies: an adaptive flood warning system, and a service discovery application. Each demo mainly consists of four stages: (1) a model of a component framework will be created and components will be associated; (2) source code of components and the XML configuration file will be generated; (3) a transition diagram will be created identifying the variants associated and reconfiguration policies will be generated; and (4) the application will be executed and some behaviour will be tested to show how the reconfigurations follow the transition diagrams while the system is running. Intentionally, some mistakes will be performed to show how the models are validated before the generation is carried out.

In particular, the case study of the adaptive flood warning system is a good example of a reconfigurable software architecture that can be dynamically updated. Therefore, we will demonstrate how new components and reconfiguration policies will be created and added during the execution of the application to show how the reconfiguration and triggers will correspond to the newly added reconfiguration policies.

5. CONCLUSIONS AND FUTURE WORK

We have presented *Genie*, a tool that offers DSLs for the design of models of component configurations and transition diagrams. These models describe the architecture of reconfigurable applications and the conditions of the environment and context that trigger the reconfiguration of the architecture. The use of DSLs promotes high levels of abstraction beyond code. Furthermore, the use of generative techniques increases the levels of efficiency, automation, and scalability. Genie has been validated by two substantial case studies.

Joint research work between researchers from Lancaster and Michigan State Universities [11, 5] has explored how transition diagram models in *Genie* can be traced from requirements models in i*. As a result, the adaptation scenarios and monitoring condition trade-offs at the requirements level were traced to the policies defining Gridkit's reconfigurations. We are exploring how to realize the automatic generation of the triggers (and their types) in the transition diagrams in Genie from the i* models. Our vision is to make requirements drivers of the reconfiguration at runtime.

Validation and the detection of conflicts between policies are needed to guarantee the correct generation of artefacts. We are enhancing Genie with validation capabilities so it can return a list of conflicts that need to be resolved.

6. REFERENCES

[1] N. Bencomo and G. Blair. Genie: a domain-specific modeling tool for the generation of adaptive and reflective middleware families. In *6th OOPSLA Workshop on Domain-Specific Modeling*, USA, 2006.

[2] N. Bencomo, G. Blair, and C. Flores. Reflective component-based technologies to support dynamic variability. In *The Second International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'08)*, 2008.

[3] N. Bencomo, P. Grace, and G. Blair. Models, runtime reflective mechanisms and family-based systems to support adaptation. In *Workshop on MOdel Driven Development for Middleware (MODDM)*, 2006.

[4] G. Coulson, G. Blair, P. Grace, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan. A generic component model for building systems software. *ACM Transactions on Computer Systems, February, 2008.*

[5] H. J. Goldsby, P. Sawyer, N. Bencomo, D. Hughes, and B. H. C. Cheng. Goal-based modeling of dynamically adaptive system requirements. In *15th IEEE International Conference on Engineering of Computer-Based Systems (ICBS 2008)*, Ireland, 2008.

[6] P. Grace, G. Coulson, G. Blair, and B. Porter. Deep middleware for the divergent grid. In *IFIP/ACM/USENIX Middleware*, France, 2005.

[7] D. Hughes, P. Greenwood, G. Coulson, G. Blair, F. Pappenberger, P. Smith, and K. Beven. 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.

[8] F. Kon, F. Costa, G. Blair, and R. Campbell. The case for reflective middleware. *Communications of the ACM*, 45(6):33–38, 2002.

[9] MetaCase. Domain-specific modeling with metaedit+.

[10] K. Pohl, G. Böckle, and F. v. d. Linden. *Software Product Line Engineering- Foundations, Principles, and Techniques.* Springer, 2005.

[11] P. Sawyer, N. Bencomo, P. Hughes, Danny andl Grace, H. J. Goldsby, and B. H. C. Cheng. Visualizing the analysis of dynamically adaptive systems using i* and dsls. In *REV'07: Second International Workshop on Requirements Engineering Visualization*, India, 2007.