2015 41st Euromicro Conference on Software Engineering and Advanced Applications

A Graphical Tool for Model-Driven Development Using Components and Services

Simone Di Cola, Cuong Tran, Kung-Kiu Lau
School of Computer Science, The University of Manchester
Manchester M13 9PL, United Kingdom
Email: dicolas,ctran,kung-kiu@cs.manchester.ac.uk

***Abstract*—Combining model-driven engineering with component-based and service-oriented approaches can potentially reap the benefits of all three approaches. In this paper we present a tool that combines these approaches. We show the key aspects of the tool and demonstrate its use with a simple example.**

I. Introduction

Model-driven Engineering (MDE) [6] is fast becoming the standard methodology for software system development [9]. The provision of tool support for MDE is also gathering pace, nowhere more so than in the Eclipse framework [7]. At the same time, component-based and service-oriented approaches (SOA) are gaining in popularity because they offer other sought-after benefits, namely high modularity and low coupling, as well as the potential to tackle scalability and complexity. Component-based approaches with properly defined underlying component models [5] are by definition model-driven.

In this paper we present a graphical MDE tool for system development that is based on a component model. In this component model, services are provided by components, and are composed when their provider components are composed. Our tool thus reaps the benefits offered by MDE, component-based and service-oriented development [1].

II. Tool Overview

Our tool supports component-based system development and its associated life cycle [3]. The latter consists of: (i) a component development phase; and (ii) a system development phase. In (i) components are designed and built, and deposited in a repository. In (ii) components are retrieved and deployed into the system under construction. Figs. 1 and 2 show our Eclipse workbench for (i) and (ii) respectively. For each development phase, the tool provides a canvas as a design space, as well as a palette of pre-defined building blocks. The tool provides continuous validation. Errors are marked in the canvas, and listed in the problems view. The tool generates code for the resulting system that can be deployed as a standalone application.

Research leading to these results has received funding from the EU ARTEMIS Joint Undertaking under grant agreement no. 621429 (project EMC2) and from the Technology Transfer Board (TSB) on behalf of the Department for Business, Innovation & Skills, UK.

III. Tool Implementation

The tool is implemented using a powerful stack of model driven technologies like Eclipse Modelling Framework (EMF) [7], Graphiti (https://www.eclipse.org/graphiti/), and CDO (https://eclipse.org/cdo/).

Our approach is based on the X-MAN component model [2]. It consists of three main entities, namely *components*, *connectors*, and *services*.

1) Components: X-MAN has two types of components: *atomic*, and *composite*. They are both fully encapsulated, i.e. they have no external functional dependencies and contain only *provided services*. An *atomic component* is a unit of computation. Its *computation unit* (CU) contains the implementation of the services it exposes. As shown in Fig. 1, according to the dragged service(s), the tool generates an interface, and an empty implementation. A *composite component* is constructed by composing pre-defined components via composition connectors.

2) Connectors: Composition connectors are (exogenous) control structures that coordinate the execution of the components they compose [4]. They are *Sequencer* and *Selector*, which provide sequencing, and branching respectively. In Fig. 2 a sequencer is shaped as an ellipse, and a selector as a rhombus. In addition, unary *adapter* connectors such as *Guard*, and *Loop* provide gating, and looping respectively. In Fig. 2 a loop is shaped as a circle, and a guard as a triangle.

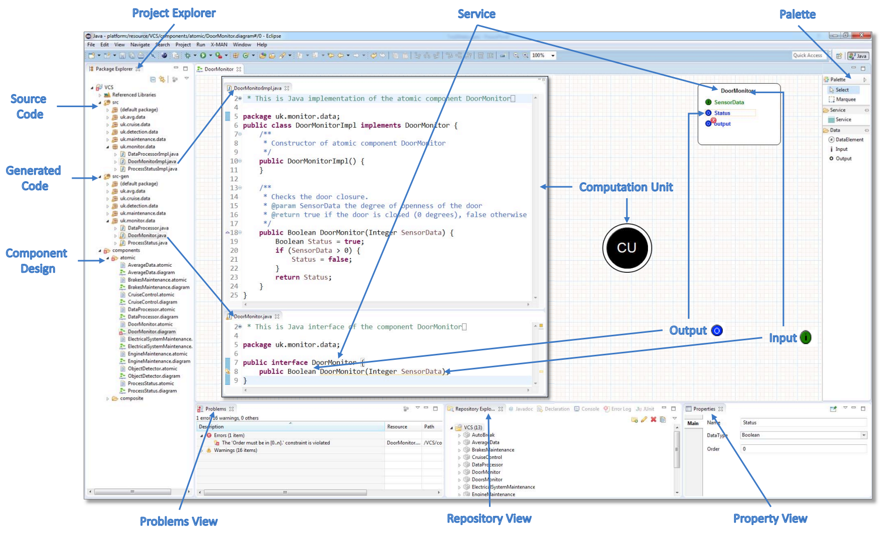
While connectors control execution among component instances, data between components flows through *data channels* (dotted arrow in Fig. 2).

3) Services: A *service* represents an operation exposed by a component. It contains two main entities: parameters, and service references. *Parameters* are inputs and outputs, while *service references* specify services in sub-components that contribute to the provided operation. In Fig. 2, a service reference is represented as a square.

Starting from the EMF meta-model, the graphical editor has been implemented using Graphiti, while the model repository is realised using CDO. Finally, we have used Xtend (https://eclipse.org/xtend) to generate code for a valid system.

IV. Discussion and Conclusion

In object-based component models (e.g. EJB) and frameworks (e.g. OSGi), services are visible at model level, but service composition is only visible at code level. Here, services



```java
 * This is Java implementation of the atomic component DoorMonitor

package uk.monitor.data;
public class DoorMonitorImpl implements DoorMonitor {
    /**
     * Constructor of atomic component DoorMonitor
     */
    public DoorMonitorImpl() {
    }

    /**
     * Checks the door closure.
     * @param SensorData the degree of openness of the door
     * @return true if the door is closed (0 degrees), false otherwise
     */
    public Boolean DoorMonitor(Integer SensorData) {
        Boolean Status = true;
        if (SensorData > 0) {
            Status = false;
        }
        return Status;
    }
}
```

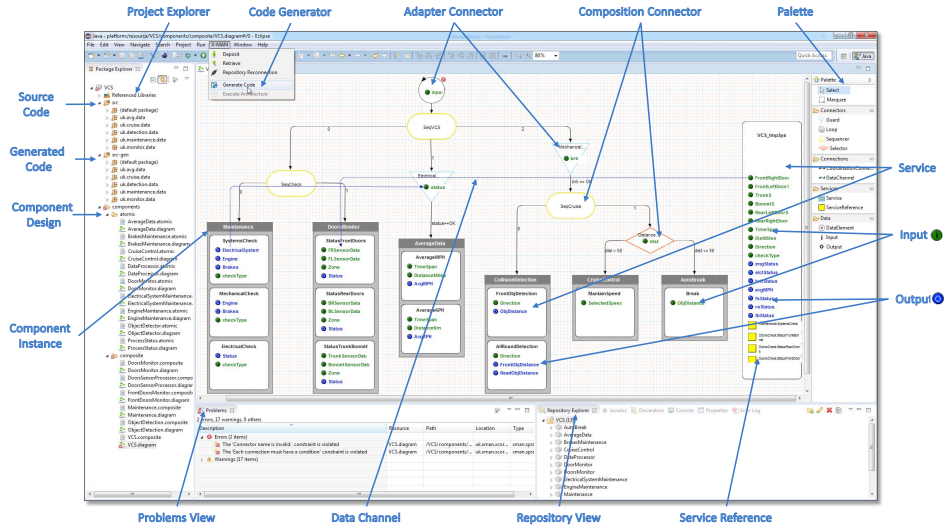
```java
 * This is Java interface of the component DoorMonitor

package uk.monitor.data;

public interface DoorMonitor {
    public Boolean DoorMonitor(Integer SensorData);
}
```

```
1 error, 16 warnings, 0 others
Errors (1 item)
  The 'Order must be in [0..n].' constraint is violated
Warnings (16 items)
```

Fig. 1: Eclipse workbench for component development.



```
2 errors, 17 warnings, 0 others
Errors (2 items)
  The 'Connector name is invalid.' constraint is violated
  The 'Each connection must have a condition' constraint is violated
Warnings (17 items)
```

Fig. 2: Eclipse workbench for system development.

are methods provided by objects, and components (objects) are composed by method calls.

In ADL-based approaches (e.g. UML 2), both services and service composition are visible at model level, but service composition is different from SOA approaches, i.e. composition is port connection rather than orchestrated.

Our approach is closer to SOA as services and service composition are visible at model level. As in web services, we use coordination for service composition. However, whilst web services use workflow languages like BPEL and BPMN [8], we define composition in the meta-model itself as composition (and adaptor) connectors.

References

[1] T. Erl. *Soa: principles of service design*, volume 1. Prentice Hall Upper Saddle River, 2008.

[2] N. He, et al. Component-based design and verification in X-MAN. In *Proc. ERTS*, 2012.

[3] K.-K. Lau, et al. Towards composing software components in both design and deployment phases. In *Proc. 10th CBSE*, pages 274–282. Springer-Verlag, 2007.

[4] K.-K. Lau, et al. Composing components in design phase using exogenous connectors. In *Proc. 32nd SEAA*, pages 12–19. IEEE Computer Society Press, 2006.

[5] K.-K. Lau. Software Component Models: Past, Present and Future. In *Proc. 17th CBSE*, pages 185–186. ACM, 2014.

[6] D. C. Schmidt. Model-driven engineering. *IEEE Computer*, 39, February 2006.

[7] D. Steinberg, et al. *EMF: Eclipse Modeling Framework*. Addison-Wesley, Boston, MA, 2. edition, 2009.

[8] S. Weerawarana, et al. *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, and more*. Prentice Hall PTR, 2005.

[9] J. Whittle, et. al. The state of practice in model-driven engineering. *Software, IEEE*, 31(3):79–85, 2014.