Towards Reusable Automation System Components

T. Aschauer, G. Dauenhauer,W. Pree

10th International Conference on Software Reuse (ICSR'2008),
Beijing, China,
May 25-29, 2008

C. Doppler Laboratory
Embedded Software Systems
University of Salzburg
Austria

Towards Reusable Automation System Components

Thomas Aschauer, Gerd Dauenhauer, Wolfgang Pree

C. Doppler Laboratory *Embedded Software Systems*, University of Salzburg
Jakob-Haringer-Str. 2, 5020 Salzburg, Austria
firstname.lastname@cs.uni-salzburg.at

Abstract. In this paper we present a domain specific language for describing an automation system, that is, its hardware and software components. These domain components form the basis of large-scale reuse so that specific automation systems can be configured efficiently.

Keywords: Component reuse, domain specific modeling language

Context and motivation

Our research group cooperates with an industry partner that is a dominant player in the area of so-called engine test bed systems that are used, for example, in the automotive industry for developing and optimizing combustion engines. Engine test bed systems are specific automation systems. Typical functions of an engine test bed system are the parameterization and visualization of its real-world components, such as the engine under test, and the sensors, as well as the measured values. The corresponding software has evolved over the last two decades and comprises about 1.5 million lines of code, mainly written in C++ and C. Originally, one of the main goals was to improve the current system's usability. A major hurdle for the users of the current system is the fact that the domain entities they have in mind (such as engines, dynamometers, measurement and conditioning devices) do not match well with the entities in its user interface. For example, a person who configures a test bed would like to deal with a graphical representation of the test bed entities, with the parameters and measured values associated with the physical components. In a typical setup there are about 10,000 parameters with about 120,000 values to be set correctly. In other words, the domain components should also be the entities the user deals with. The domain components should be reusable assets that allow an efficient configuration of a specific engine test bed system. For that purpose we defined the *Domain Component Description Language* (DCDL) sketched in the next section.

Domain Component Description Language (DCDL)

DCDL is a domain specific language for describing the components of a test bed automation system, either as text or in an equivalent visual form. For the sake of brevity we use the textual representation below.

The primary entities of DCDL are components which describe the structure and the behavior of automation system components such as the engine under test or measurement devices, but also the relevant properties of an automation system's software components. The electric current is an example of a physical component property.

The explicit description of hardware and software properties is required to check the validity of compositions and thus to ensure the consistency of the DCDL model.

Consider, for example, a temperature sensor and a measurement device. The sensor's DCDL description comprises its plug's shape and its emitted electrical signal. The measurement device's description also comprises its plug's shape and its accepted electrical signal. If the sensor and the measurement device are connected in the test bed description, a validity check is performed that only allows the sensor to be plugged into the measurement device if both the plug shape and the electrical signals match.

Viewing different component aspects. The various aspects of a DCDL component can be grouped and component editors typically show them in separate views:

- *Physical View*: Represents physical and if applicable electrical aspects such as plugs and wires (see figure 1).
- *Functional View*: Represents functional aspects, such as PID controllers and limit monitors. This view is similar to dataflow modeling languages such as Simulink [1].
- *Parameter View*: Represents variability aspects in terms of name/value pairs, e.g. plug shape descriptions or PID controller values.
- *Operation View*: Visualizes a component during the operation of the test bed automation system, for example, by showing relevant measurement values and changing their color in case they are not within predefined limits.

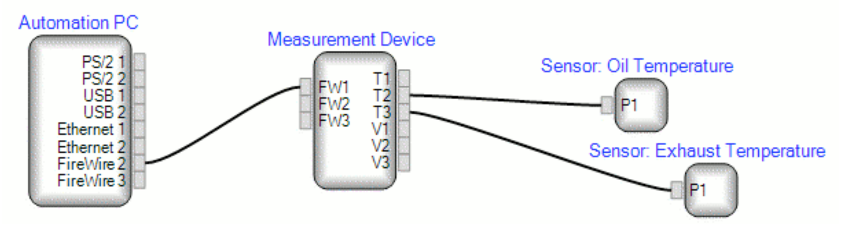

Fig. 1. Sample physical view of a test bed

DCDL component definition by example. We use a table format to illustrate the definition of DCDL components (see figure 2). The table shows two simplified *Engines*, E_1 and E_2. Possible properties of engine components are specified in columns. It is not mandatory to define values for each property. E_2, for example, does not specify the property *Ignition*.

Category: Engine			
Component	*Cylinders*	*Inertia*	*Ignition*
E_1	8	1.06 kgm^2	Plug 15
E_2	6	1.04 kgm^2	

Fig. 2. Two sample DCDL components

DCDL offers mechanisms to reuse component definitions. Interviews with domain experts showed that for them copying and pasting a component definition is a natural way of reuseing it. Therefore DCDL supports what is called prototypical inheritance in the object-orientated programming paradigm [2]. Figure 3 shows an engine E_3, which is defined by copying the definition of engine E_1. The properties *Cylinders* and *Ignition* are inherited and their values are unchanged, the property *Inertia* is inherited but its value was changed, and a new property N_{max} is added.

Category: Engine				
Component	*Cylinders*	*Inertia*	N_{max}	*Ignition*
E_1	8	1.06 kgm^2		Plug 15
$E_3 \leftarrow E_1$	*8*	1.05 kgm^2	12,000 rpm	*Plug 15*

Fig. 3. Component extension via prototypical inheritance

DCDL-based composition. DCDL was designed so that the compatibility of automation system components can be checked. When components are reused to assemble a specific test bed system, the DCDL type system ensures that users can define only valid compositions. The following concepts form the backbone of the type compatibility check:

- Components are assigned to user defined categories. The type system treats components of different categories as incompatible. Since all components in figures 2 and 3 are classified as *Engine*, they are of the same category and are, for example, not compatible to any component of another category such as *Measurement Device*.
- Every view defines a separate type system. The physical view, for example, defines compatibility between plugs in terms of plug shape. The parameter view's type system is analogous to that of imperative programming languages with strong type checking such as Java.

Abstract DCDL components. DCDL offers the possibility to define types and values of properties as *unspecified*. Components with at least one unspecified property are *abstract components*. A valid DCDL model must not contain abstract components. Abstracting from concrete components should further support the reuse of component definitions. The properties of abstract components are listed, but not yet typed. The following example shows how an abstract engine *AbstractEngine* could be modeled.

```
COMPONENT AbstractEngine CATEGORY 'Engine'
  Nmax : UNSPECIFIED := UNSPECIFIED
END
```

Hierarchical composition. DCDL components can be hierarchically composed of other components. The dynamometer example shows how a dynamometer *D* and an abstract test bed *AbstractTestBed* are defined. *AbstractTestBed* is composed of *D* and *AbstractEngine*. Since *AbstractEngine* is an abstract component, *AbstractTestBed* is also abstract. The property *Nmax* of engine *AbstractEngine* is used to express the constraint that only engines with a lower maximum rotation speed than the dynamometer may be mounted. Although unspecified properties may be used to express constraints, these constraints can not be enforced until all referenced properties are fully specified in components that are based on abstract components.

```
COMPONENT D CATEGORY 'Dynamometer'
  Nmax : REAL := 20000[rpm]
END

COMPONENT AbstractTestBed CATEGORY 'Test bed'
  Engine : AbstractEngine
  Dyno : D
  Dyno.Nmax >= Engine.Nmax
END
```

From abstract to concrete components. Finally we illustrate the transformation of an abstract component to a specific one. First, an engine *E1* is defined whose properties have the same names as the ones of *AbstractEngine*. Second, *SampleTestBed* is created as a clone of *AbstractTestBed*. *SampleTestBed* redefines the property *Engine* by replacing the component *AbstractEngine* with *E1*. *SampleTestBed* is a concrete component, since it neither contains abstract components nor does it contain unspecified properties itself.

In the context of *SampleTestBed*, *AbstractEngine* is substitutable by *E1* since a) both components are of category *Engine*, b) both components have a property *Nmax* and c) *D.Nmax* has the same data type and unit as *E1.Nmax*. All parameters are specified and thus the constraint inherited from *AbstractTestBed* can be enforced.

```
COMPONENT E1 CATEGORY 'Engine'
  Nmax : REAL := 12000[rpm]
END

COMPONENT SampleTestBed LIKE AbstractTestBed
  Engine : E1
END
```

Related Work

There is a trend in the embedded industry to explicitly describe the overall computing infrastructure, the static hardware and software setup as well as the interactions between the components. One example is AUTOSAR [3], an initiative by the automotive industry. As automation systems differ from automotive computing platforms, the description differs, though the vision is similar. Modeling languages such as UML and SysML [4] could be harnessed for the visual representation of CDCL.

References

1. The MathWorks Simulink, www.mathworks.com/products/simulink
2. Abadi, M. and Cardelli, L.: A Theory of Objects. Monographs in Computer Science, Second Edition, Springer-Verlag, New York (1998)
3. AUTOSAR (an acronym abbreviating AUTOmotive open System ARchitecture) see www.autosar.org
4. Object Management Group: OMG Systems Modeling Language (OMG SysML™), Unified Modeling Language (UML), www.omg.org