Applying CAESAR to software product lines in the context of hardware devices

Egon Wuchner
Siemens AG
Otto-Hahn-Ring 6
81739 Munich, Germany
egon.wuchner@siemens.com

ABSTRACT
Features of complex hardware devices like a magnetic resonance machine often stem from new hardware capabilities involving the exchange or addition of a whole set of hardware units and their software. Product lines of such systems can be organized along the lines of such system families of collaborative hardware units and their software. Additionally, health devices have to comply with strict regulations in order to prevent any danger for a patient. Thus, amending a system family or adding one in order to cover a new feature has to guarantee the installation of allowed combinations only. This paper tackles the challenges of software product lines for hardware devices from an architectural/design point of view. We provide a solution approach by using CAESAR, a new aspect-oriented language. We identify certain restrictions of CAESAR and propose to extend its binding mechanism.

1 INTRODUCTION AND EXAMPLE

Product lines and/or system families of software are normally discussed in the context of features. Vendors of software systems and tools should be enabled to compose and price their software according to the specific needs of a customer or market. Additionally, there is another source of software product lines with its own facets, difficulties and challenges. For example vendors of medicine devices like a high-end magnetic resonance or a computer tomography machine have to combine a bunch of hardware units which has to be controlled and operated by the corresponding software system.

A new feature often stems from new hardware capabilities, e.g. a rotating patient table that supports a magnetic resonance machine to compute a new kind of patient images. But the software modifications are not restricted to the table hardware unit and the algorithmic part computing the images only. Either other hardware units like a display already existent in the system are involved and have to be replaced in order to show the degree of rotation. Or new units have to be added to the system.

The example of a rotating table is simple but totally sufficient to demonstrate the software challenge of different combinations of hardware units supporting a feature. Such a new table has implications on other hardware units. For instance the patient lying in the tube of a magnetic resonance might sense a rotating tube as scaring since it partially covers the entry and exit of the table into the tube. Thus it requires light and ventilation of different levels inside the patient tube.

In addition the old display showing the table position has to be adapted in several ways. The new position of the third dimension or the degree of rotation has to be displayed, too. Furthermore the current ventilation and light levels have to be included into the display as well. Next table gives an overview of old and new hardware devices by specifying the respective classes, their responsibilities and collaboration with each other.

Class	***Responsibility***	***Collaboration***
old table	moving the physical table in x, y directions, providing information about the current position of both directions.	with the display to show the table positions in x, y directions by notification.
old display	displaying x, y positions of the table	-
new table	old table plus moving the physical table in a third direction (z) and providing information about this directional position.	with ventilation and light: switching it on/off with rotating start and fine-tuning the ventilation/light with increasing rotation. with new display to display the z-position of the table by notification.
ventilation/ light	switching the physical devices on/off, level of ventilation/light can be set	-
new display	displaying the z position, displaying the level of ventilation/light	-

The new hardware configuration of a rotating table consists of a rotating table, new ventilation/light devices and a replaced display. But there are other valid combinations of old and new hardware as well. For example a customer might wish a rotating table in order to harness the new measurement capabilities. But he does not insist on a new display since he considers it sufficient to check the new position, the ventilation and light manually, by sight or noise. On the other hand a new table makes ventilation and light obligatory. Next table separates these combinations into recommended, possible and forbidden ones:

Recommended			
new table	vent./light	new display	new configuration includes all
"	"	old display	less expensive but new configuration, doctor has to check Vent/light by sight and noise
Possible			
old table	no vent./light	old display	old configuration
"	"	new display	no value-added
"	vent./light	old display	luxery, but possible
"	"	new display	even more luxery, but possible
Forbidden			
new table	no vent./light	old display	forbidden since it might endan-ger the patient
"	"	new display	does not make sense

Relying on this example we go on with Section 2 stating the problems subsequent sections aim at. Section 3 provides a solution approach using CAESAR, a new aspect-oriented language allowing to express and to specialize abstract collaboration interfaces (ACIs) supporting the notion of family polymorphism. The last section explores specific shortcomings of CAESAR which have shown up during the elaboration of our solution approach. Finally, it suggests possible extensions to CAESAR in order to overcome these drawbacks. Concluding Remarks summarize the benefits of this approach.

2 PROBLEM STATEMENT

The table control example from the introductory section can be considered as a system family. Each software part specific to an affected hardware unit of the table control family has to reflect these new or changed capabilities. But the challenges with respect to the software of such a hardware control system family go beyond and are manifold. In general, the collaboration between these new hardware units and their cooperation with involved and kept hardware units has to be adapted in a compliant way. For instance, a new table class can not be reused in combination with an old display since it requires the display to be able to receive "display ventilation" requests.

Except for totally new hardware configuration fully supporting a new feature several other hardware combinations seem feasible and reasonable. These half-old, half-new configurations partially include old hardware units and a set of new ones providing the new feature but with some restrictions (e.g. concerning its convenient operation).

Take the two recommended members of the table control system family. Even these two related members require a different implementation of the new table class despite providing the same interface. The first family variant of table control contains a new table activating light, ventilation and displaying their operational level on the new display during a rotation. Compared to this scenario the new table of the other recommended variant interacts with ventilation and light without notifying the old display of these activities. Generative approaches are certainly able to deal with such variations but following additional challenges have to be met:

Possible, but dangerous mix of types from different variants of a system family: excluding the generation of forbidden variants by mistake does not suffice. Furthermore, it is essential to exclude mixing types of one family member with types of another one coincidentally. Let us keep to both recommended variants again in order to illustrate this point. Using the new table implementation of the second variant within the context of the first one can hardly be prevented at compile-time since they only differ in their implementation. But it leads to table control misbehaviour because of lacking notifications of this new table implementation requesting the new display to show ventilation, light and rotational activities. Keeping in mind that health-devices are subject to very strict regulations, mixing types of different family members might have unpredicted impact on a patient.

On-site changes without a reinstallation of the whole system: customers already running a magnetic resonance machine wish the new hardware units and the software to be introduced to their machines on site. Consequently, the table control system family and other family instances represent the right granularity of such partial deployments

without a restart. Thus, guaranteeing the installation of consistent variants only is even more important.

3 SOLUTION APPROACH WITH CAESAR

CAESAR ([1], [2]) allows to define abstract collaboration interfaces (ACI) containing several nested interfaces interacting with each other to a high degree. For instance all classes of a new configuration like table, ventilation, light, display addressing this special issue of table control can be captured as nested interfaces within such a collaboration interface. A system family can be mapped to a collaboration interface and derived collaboration interfaces might capture the variants of it.

Furthermore, CAESAR requires a binding class specifying how an implementation of a collaboration interface is to be bound to the rest of the software system. Within the context of the presented problem, a collaboration interface of a system family variant is accompanied by a binding to the old table and display classes of an existent and running MR system.

Besides its support of seamless integration of design results like ACIs and their implementation on code level, CAESAR provides the feature of family polymorphism. This mechanism guarantees that no nested interface implementation instances of different ACIs are able to interact. Only implementations of interfaces of the same ACI can work together, thus building a family of collaborative instances. Consequently, types and instances of different family variants (recommended, possible and forbidden) can not be mixed by mistake. By the way different variants of a family are already expressed on design and code level.

Last but not least, another advantage of CAESAR is its "on-demand modularization" feature which supports doing the binding of an existent system with a certain ACI implementation on demand. Hence, doing deployment of a new configuration without a complete software update seems feasible.

Applying CAESAR

A solution approach has to start with an answer to the question how to express a system family of old and new configurations of table control in the software. This section assumes an existing system containing a non-rotating table and its normal display. This system (Table and Display) is the base of further configurations and can be considered as already delivered to customers.

```
package MrSystem;

class Table {
  Table( Display display) {this.display=display};

  int moveX(int xPos) {…;
    this.display.displayX(xPos); …
  };

  int moveY(int yPos) {…;
    this.display.displayY(yPos); …
  };
  …
}

class Display {
  …
  void displayX(int xPos) {…};
  void displayY(int yPos) {…};
}
```

A new configuration of table control involving a rotating table, a new display, ventilation and light control can be expressed in a modularized way within an abstract collaboration interface called TableControl.

```
collaboration interface TableControl {
  provided Display getDisplay();
  provided VentControl getVent();
  provided LightControl getLight();

  interface Table {
    expected int moveX( int xPos );
    expected int moveY( int yPos );
    provided int rotate( int degree );
  }

  interface Display {
    expected void displayX( int xPos );
    expected void displayY( int yPos );
    provided void displayRotating( int degree );
    provided void displayVent( VentLevel level );
    provided void displayLight(LightLevel level );
  }

  interface VentControl {
    provided void switchOn();
    provided void switchOff();
    provided void setLevel(VentLevel level);
    provided VentLevel getLevel();
  }
  // the same about LightControl
}
```

The nested interface Table and Display contain expected method declarations. These methods aim at binding the new configuration to the existent table control of the MR system.

```
class TableControl_NewImpl provides TableControl
{
  final this.Display display = new this.Display();

  Display getDisplay() { return this.display; }

  … // the same about getVent and getLight

  class Table
  {
    …
    int rotate( int degree ) {
      …; // rotate physical table
      this.getDisplay().
        displayRotating( degree );

      this.getVentControl().switchOn();
      if( degree>… ) this.getVent().setLevel(…);

      // the same about Light
```

```
    }
  } // end class

  …// Display, Vent and Light
}
```

The implementation of TableControl provides an implementation of the nested interfaces. The sketched code also illustrates the collaboration between the nested interface implementations of the ACI and how they use their extra functionality. For instance `Table.rotate` interacts with the new display and the new vent/light control.

Furthermore note the exploitation of family polymorphism. A rotating table can only be created by using an instance of TableControl_NewImpl. This table implementation only cooperates with display, Vent, light instances of the same ACI by specifying their types in the context of a TableControl_NewImpl instance only. (By the way these instances have to be singletons indicated by the usage of final field members). The binding of this implementation to the existent system looks like the following:

```
class TableControl_NewConfig binds TableControl
{
  class Table binds TableCotnrol.Table
  wraps MrSystem.Table
  {
    private final MrSytem.Table table;
    …
    int moveX( int xPos ) {
      // implicit call of Display.displayX
      this.table.moveX( xPos );

      this.getVent().switchOn();
      if( xPos>… ) this.getVent().setLevel(…);

      … // the same about Light
    }
  } // end class

  … // binding of Table

  … // classes Vent and Light need no binding
}
```

The binding has to address the collaboration with the initial Table and Display classes. In contrast the purpose of expected methods being called within the implementation of the corresponding ACI we use the expected methods to extend the implementation of their counterparts within the initial Table and Display methods. The necessity of this unusual step stems from the fact that the `MrSystem.moveX` and `MrSystem.moveY` have to be kept in order to keep their client code untouched. Hence their implementations have to interact with the new Vent and Light classes.

4 PRODUCT LINES

The last section presumed an existent system of a certain base configuration. But designing software of product lines has to take this base configuration into account from the very beginning.

The first step in developing the table control software for different variants could be to analyze commonalities of all combinations and their variation points. For instance the commonality of all table control system family could be a non-rotating table and its corresponding display. Mapping the results of such an analysis to CAESAR could mean to modularize the common part of any combination into a base ACI (plus an implementation) and capturing the variable configurations into different derived ACIs (whose implementations can be derived from the implementation of the base ACI implementation).

This procedure preserves all the advantages that lead to the selection of CAESAR in order to find a solution approach. But it has some negative implications. Since the table control system family represents only a small portion of the system software and the underlying hardware making up a magnetic resonance machine we have to consider other hardware control collaborations as well. An MR measurement implies its own hardware parts being specific to measurements (e.g. coils). Besides the measurement procedure also interacts with the table control because a measurement implies moving the table into certain positions. In addition, combinations of measurement control and table control configurations can also be separated into recommended, possible and forbidden categories.

The first thought how to deal with the combinations of a measurement and table control in CAESAR is to combine the base configurations of table and measurement control into an ACI of its own (let us call it Measurement-TableControl). Note that the base configuration of table is an ACI itself. The same holds true about the measurement control configuration. Each variable configuration of measurement and table control have to be captured within an own ACI derived from MeasurementTableControl.

Taking this step further results in a product line tree of derived and nested ACIs with varying hardware control ACIs as leafs. Thus, selecting and assembling a complete magnetic resonance machine means to select a sub-tree containing a leaf for each hardware control family. The drawback about this idea is the combinatorial explosion of ACIs (and their implementations) when composing the software of a magnetic resonance machine. Each ACI combination needs a special binding. The exclusion of forbidden combinations reduces the number of necessary bindings only slightly.

REFERENCES

[1] M. Mezini, K. Ostermann. *Integrating independent components with on-demand remodularization.* In Proceedings of the 17th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, 2002

[2] M. Mezini, K. Ostermann. *Conquering Aspects with Caesar.* In Proceedings of the 2nd International Conference on Aspect-Oriented Software Development, pages 90-100, 20