Handling the Dynamic Reconfiguration of Software Architectures using Aspects

Cristóbal Costa-Soria[1], Jennifer Pérez[2], José Ángel Carsí[1]
[1]*ISSI, Dept. of Information Systems and Computation,*
Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain
[2]*Escuela Universitaria de Informática,*
Technical University of Madrid (UPM), Ctra. Valencia km. 7, 28051 Madrid, Spain
ccosta@dsic.upv.es, jenifer.perez@eui.upm.es, pcarsi@dsic.upv.es

Abstract

Currently, most software systems have a dynamic nature and need to evolve at run-time. For this reason, the dynamic reconfiguration of software architectures is a challenge that must be dealt with to enable the creation and destruction of component instances and their links at run-time. This challenge is even greater when there are autonomous composite components which also need reconfiguration capabilities to evolve their internal compositions. This paper presents a novel approach to dynamically reconfigure software architectures taking advantage of aspect-oriented techniques. The approach presented is a platform-independent alternative whose aim is to increase the reuse and to decrease the maintenance effort. It deals with the challenge of reconfiguring composite components that: (1) are easy to maintain, since the dynamic reconfiguration concern is separated from the other concerns; (2) can autonomously reconfigure themselves, since each composite component is provided with dynamic reconfiguration services to change its internal architecture.

1. Introduction

The development of self-organizing software systems is a great challenge, since they are conceived to help the management of large systems, which may consist of heterogeneous entities, and with a lot of interactions with other software components. Software architectures [10] provide techniques for describing the structure of complex software systems. This structure is made up of architectural elements (components and connectors) and their interactions with each other. Software architectures have a hierarchical structure where components can be composed into complex components and these, in turn, can be composed into more complex components. To enable a software architecture to be self-organized, i.e. to be capable of reconfiguring their topology by itself at run-time, first its dynamic (re)configuration has to be supported. Thus, the system will be able to create or destroy their architectural element instances and links at run-time. Moreover, reconfiguration capabilities must be also provided to complex components, in order to evolve their internal compositions in an autonomous way.

Programmed, dynamic reconfigurations [6] can be used to describe self-organizing software architectures. Programmed reconfigurations are pre-planned changes (i.e.: described at design time) which are triggered at run-time by the system itself. This kind of reconfiguration is usually specified using an Architecture Description Language (ADL), so the language constructs (e.g. architectural elements) can invoke the reconfiguration capabilities. However, one of the disadvantages of using programmed reconfigurations is that they crosscut the system functionality, making both the programmed reconfigurations and the system functionality difficult to maintain. Another disadvantage of the programmed reconfiguration is the fact that it is impossible to predict all the programmed reconfigurations that will be necessary.

In this paper, we are going to address the first problem: aspect-oriented techniques are used to avoid the entanglement between reconfiguration descriptions and system functionality. The approach presented is a platform-independent alternative while increasing reuse and decreasing the maintenance effort.

2. The Dynamic Reconfiguration Concern

In the literature, dynamic reconfiguration has been provided to software architectures in two main ways

[2]. On the one hand, some approaches provide dynamic reconfiguration through a global, specialized entity [7] that describes all the reconfiguration actions of the whole software system. This entity is in charge of changing the software architecture at runtime. One of the main disadvantages is that it tangles the reconfiguration actions of different composite components. Each composite component may need different reconfigurations, which should be described and executed independently of each other. Another disadvantage is the fact that it centralizes all the reconfiguration needs. As a result, when there is a high number of reconfiguration requests, this entity is turned into a bottleneck. It is not able to concurrently process the requests and the evolution performance is reduced. On the other hand, other approaches have used a decentralized approach [1]. They have introduced a set of generic reconfiguration statements (i.e. *create*, *destroy*, *attach*, *detach*, etc.) in the ADL to enable each component to reconfigure the architecture. However, this has the disadvantage that the reconfiguration statements are often tangled with the component functionality. Some approaches [8] have addressed this by isolating evolution from functionality in different processes. However, the communication among these processes introduces dependencies which decreases reuse and maintainability.

Aspect-Oriented Software Development (AOSD) proposes the separation of the crosscutting concerns of software systems into separate entities called aspects. This separation avoids the tangled concerns of software, allowing the reuse of the same aspect in different entities of the software system as well as its maintenance. The application of AOSD techniques in software architectures improves their evolution and flexibility since the crosscutting concerns of the architecture are not tangled, and the components and their interactions can be adapted easily by adding or removing aspects.

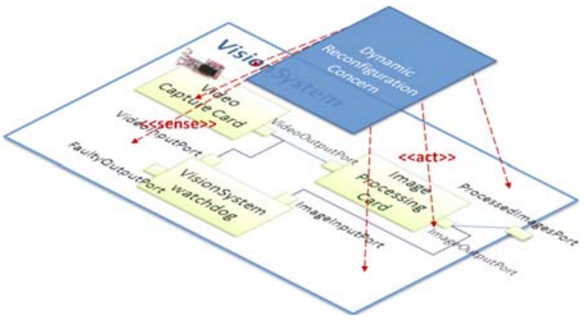

Figure 1. The Dynamic Reconfiguration concern

Dynamic reconfiguration can be observed as a crosscutting-concern of a flexible, reconfigurable software architecture. Dynamic reconfiguration is an architectural concern, because it acts on the configuration of a concrete architecture by creating or destroying component/connector instances and links at runtime. For this reason, this concern should be isolated from the functionality of architectural elements, in order to improve the maintenance and reuse of both reconfiguration code and functional code. The dynamic reconfiguration concern should allow us: (i) to define how and when the architecture must be reconfigured, that is, to define programmed reconfigurations; and (ii) to provide the necessary mechanisms to inspect and reconfigure the architecture at runtime.

Several proposals for the integration of aspects in software architectures have emerged [4]. PRISMA [9] is a proposal that integrates AOSD and Component-Based Software Development to describe software architectures of complex software systems at a technology independent level. In PRISMA, architectural elements are specified by importing aspects. This approach is applied to the PRISMA model, because its Aspect-Oriented ADL (AOADL) allows to describe aspect-oriented software architectures in a simple way. In PRISMA, the Dynamic Reconfiguration concern has been encapsulated in two aspects, called *ReconfigurationAnalysis* and *Configuration*.

2.1. The ReconfigurationAnalysis Aspect

In order to keep reconfiguration specifications (i.e. programmed reconfigurations) from being scattered with the rest of the composite component functionality, they have been encapsulated in an aspect called *ReconfigurationAnalysis* (for briefity, RA). This aspect describes *when* and *how* reconfigurations must be executed: (i) it defines the stimulus (e.g. external or internal events, properties, etc.) that trigger reconfigurations, by means of reconfiguration triggers; and (ii) it specifies how a composite component can be reconfigured, by means of configuration transactions.

On the one hand, a reconfiguration trigger is a set of conditions which, if true, activates one or more configuration transactions. The condition may check any property of the architecture, or if any particular event has been raised. A RA aspect can include one or more reconfiguration triggers.

On the other hand, a configuration transaction is a set of ordered atomic reconfiguration operations that must be executed in a transactional way (all or none) to obtain a new configuration. However, these reconfiguration operations are not generic (i.e. *create*, *destroy*, *attach*, *detach*, etc.), like other approaches do

[1],[5],[6]. The use of unconstrained, generic reconfiguration operations can lead the architecture to the *configuration erosion* problem [2], which takes place as a result of successive uncontrolled reconfigurations on the architecture. In order to tackle this problem, our approach limits the set of available reconfiguration operations, and allows to set specific reconfiguration constraints. On the one hand, reconfiguration is limited by the constraints defined in the architecture pattern each PRISMA composite component defines [3]. By using this information, the generic reconfiguration operations (i.e. create, attach, bind, etc.) are specialized for each component type and connection that are allowed in the composite component architecture. For instance, the reconfiguration operation named *createImageProcSW* allows to create instances of the component named ImageProcSW; the reconfiguration operation named *attachImageProcSW_VideoCapture* allows to connect ImageProcSW and VideoCapture component instances. On the other hand, specific reconfiguration constraints can also be defined. Since the headings of the specialized reconfiguration operations are made available to the software architect, the operations can be extended to include additional constraints or operations. For instance, some property of a component can be checked before its destruction, or a message can be sent each time an *attach* operation is executed.

The RA aspect is imported by each composite component of the software architecture that needs programmed reconfigurations. Then, the skeleton of the RA aspect is automatically generated for the composite component it is weaved to. It provides the set of available, specialized reconfiguration operations. Thus, the software architect can specify the different configuration transactions, by using the specialized reconfiguration operations, and the reconfiguration triggers that activate these transactions. However, the reconfiguration transactions defined can only modify the composite component while mantaining its architecture consistence.

Figure 2 shows a fragment of the RA aspect, using the PRISMA AOADL, which specifies how to dynamically reconfigure a composite component (a computer vision subsystem of a robot) when a fault event ("faultyOuput") is raised. When this event is received, the reconfiguration trigger (see Figure 2, nº1) activates the configuration transaction named *RepairImageProcessingUnit*. In PRISMA AOADL, the specification of transactions is preceded by the reserved word **Transactions**. A transaction consists of a set of subprocesses that coordinate a set of services (i.e. reconfiguration operations), which are specified and synchronized with each other using a π-calculus derived notation. Each one of the services is executed atomically; if the execution of one of them fails, the execution of other services already executed is undone. The configuration transaction fragment shown in Figure 2 first gets the references (concrete IDs) to the component instances of each component type involved in the reconfiguration process (see nº2). Then (see nº3): (i) a new component instance is created, to replace the faulty one; (ii) the new instance is attached to the component instance connected to the old one ("videoCID", an instance of the VideoCapture component); and (iii) the attachments of the faulty instance are removed.

```
ReconfigurationAnalysis aspect VisionSysRecAnalysis
  ...
  [headers of specialized reconfig. operations]
Triggers
1)  REPAIRIMAGEPROCESSUNIT when
        Event?("faultyOutput",{"ImageProcCard"}):1
Transactions
  in RepairImageProcessingUnit():
2)  BEGIN::= getVideoCapture(videoCID) -->
      getImageProcCard(oldImPrID) -->
        [more calls to obtain references
         to component instances]  --> RECONF;
3)  RECONF ::= createImageProcSw({},newImPrID) -->
      attachImageProcSw_VideoCapture(newImPrID,
           videoCID, attID) -->
      detachImageProcCard_VideoCapture(oldImPrID,
           videoCID)  -->
    [more reconfig. operations];
End_Aspect VisionSysRecAnalysis;
```

Figure 2. Fragment of a ReconfigurationAnalysis aspect.

2.2. The Configuration aspect

The *ReconfigurationAnalysis* aspect describes at an high abstraction level *when* and *how* an architecture should be dynamically reconfigured. However this aspect only specifies the reconfiguration process, but it does not provide the reconfiguration mechanisms to perform these operations. For this reason, the dynamic reconfiguration mechanisms have been encapsulated in a different aspect, the *Configuration* aspect. This aspect and the *ReconfigurationAnalysis* aspect are imported together to provide a composite component with dynamic reconfiguration capabilities. In fact, the *Configuration* aspect is the provider of the reconfiguration mechanisms used by the *ReconfigurationAnalysis* aspect.

The *Configuration* aspect encapsulates every property and behaviour related to dynamic reconfiguration. This aspect contains the configuration of the composite component architecture by means of a set of attributes, and provides a set of services in order to maintain and evolve this configuration. On the one hand, the attributes store information about the configuration state of the architecture. The main ones (among others not described here for space reasons) are these that store: (i) references to each architectural element instance of the composite component; and (ii) references to each created attachment (i.e. established connections among components and connectors) of the composite component.

On the other hand, the *Configuration* aspect provides two kind of services, which let to query and modify the configuration state stored in the attributes: *introspection services* and *reconfiguration services*. *Introspection services* allow a composite component to know, at run-time, how is itself configured. The main ones (among others) are: *getArchElements* and *getAttachments*, to get all the IDs (i.e.references) of the architectural element instances and attachments that are currently established in the composite component; *getArchElement* and *getAttachment* provide information about a concrete component or attachment, by providing its ID. *Reconfiguration services* allow a composite component to modify its configuration, that is, reconfigure its architecture. Examples of these services are *newInstance*, *destroyInstance*, *addAttachment*, *removeAttachment*, etc.

The implementation of these reconfiguration services is technology dependent, and its details are not described here for space reasons. However, the contribution is that the *Configuration* aspect hides inside the technology dependent details, and the reconfiguration services that it provides are generic and technology independent. Thus, the architecture specifications (and reconfigurations) are unaware of changes on the underlying technology. In this way, the *Configuration* aspect is developed only once for a target platform; then it will be able to be reused in different software architectures executed in the target platform. Likewise, the *ReconfigurationAnalysis* aspect is specified once for each software architecture; then it will be able to be used in different platforms.

3. Conclusions

This paper has presented an approach to dynamically reconfigure software architectures taking advantage of aspect-oriented techniques. Tangled and scattered code is avoided: (i) the reconfiguration concern is described separately from the functional concern, so maintenance of both concerns are improved; (ii) reconfiguration specifications (described in the *ReconfigurationAnalysis* aspect) are described independently of the reconfiguration mechanisms, so reconfigurable architecture descriptions are technology independent; and (iii) the reconfiguration mechanisms are encapsulated in a single aspect (the *Configuration* aspect) which provides to each composite component its own reconfiguration mechanisms to reconfigure itself independently of other composite components of the system. In the future, we will show a case study to validate our approach, and a tool to manage dynamic reconfigurations from a platform independent perspective, while improving reuse and maintenance.

Acknowledgements. This work is cofunded by the Spanish Dept. of Science & Technology, under the National Program for Research, Development and Innovation, META project TIN2006-15175-C05-01, and by the European Regional Development Fund. It is also partially funded by the Univ. Politécnica de Valencia under the project "Quality-Driven Model Transformations", and by the Com. de Madrid and the Rey Juan Carlos Univ. under the project URJC-CM-2007-CET-1555. The work performed by C.Costa-Soria is supported under the contract BFPI06/227 of the Conselleria d'Educació i Ciència (Generalitat Valenciana).

4. References

[1] Batista, T., Joolia, A., Coulson, G.: Managing Dynamic Reconfiguration in Component-Based Systems. 2nd Eur. Workshop on Software Architectures (EWSA'05). LNCS, vol. 3527. Springer (2005)

[2] Bradbury, J.S., Cordy, J.R., Dingel, J., et al., "A Survey of Self-Management in Dynamic Software Architecture Specifications". Workshop on Self-Managed Systems (WOSS'04). California, 2004.

[3] Costa-Soria, C., Pérez, J., Carsi, J.A., "Managing Dynamic Evolution of Architectural Types". In 2nd European Conf. on Software Architecture (ECSA'08). LNCS, vol. 5292. Springer, 2008.

[4] Cuesta, C.E., Romay, M.d.P., Fuente, P.d.l., et al., "Architectural Aspects of Architectural Aspects". In 2nd European Workshop on Software Architecture (EWSA'05). LNCS, Vol. 3527. Springer, 2005.

[5] David, P., Ledoux, T., "An Aspect-Oriented Approach for Developing Self-Adaptive Fractal Components". In 5th Int. Symp. on Software Composition (SC'06). LNCS, Vol. 4089. Springer, 2006.

[6] Endler, M., & Wei, J., "Programming Generic Dynamic Reconfigurations for Distributed Applications". 1st Intern. Workshop on Configurable Distributed Systems. London, UK, 1992.

[7] Garlan, D., Cheng, S., Huang, S., et al., "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure". *Computer*, vol. 37, IEEE, 2004.

[8] Morrison, R., Balasubramaniam, D., Kirby, G., et al, "A Framework for Supporting Dynamic Systems Co-Evolution". *Autom. Soft. Eng*, vol.14(3). Springer, 2007.

[9] Pérez, J., Ali, N., Carsí, J.A., Ramos, I., et al., "Integrating aspects in software architectures: PRISMA applied to robotic tele-operated systems". *Information & Software Technology*, vol. 50(9-10). Elsevier, 2008.

[10] Perry, D., & Wolf, A., "Foundations for the Study of Software Architecture". *ACM SIGSOFT Software Engineering Notes*, vol. 17(4), 1992.