THE DESIGN OF AN EGOVERNMENT APPLICATION USING AN ASPECT ORIENTED SOFTWARE ARCHITECTURE

Djamal BENNOUAR
Saad Dahlab University
09000 Blida, Algeria
dbennouar@gmail.com

Abberrezak Henni
INI, Oued Smar
16000, Algiers, Algeria
henni@ini.dz

Abdelfettah saadi
CDTA,Baba Hassen,
16000, Algiers, Algeria
asaadi@cdta.dz

ABSTRACT

Software development using software architecture approaches and aspect oriented programming represents today a very promising way for the design of high quality software at lower costs. The Integrated Approach to Software Architecture (IASA) is an Aspect Oriented Software Architecture Approach using a component model totally independent from any software mechanism, mainly the interface concept. The IASA component model provides facilities not supported by nowadays software architecture tools to easily specify any topology an architect can imagine. It is used here to show how it is easy to design at a high level of abstraction, an EGovernment application using an Aspect Oriented approach.

Keywords: Software Architecture, Component, Port, Connector, Aspect, E-government

1 INTRODUCTION

After the great success of the object model in various fields of software system design (RAD, Distributed Object Infrastructure, Component models such as EJB, CCM, .Net etc.) the software engineering has known these last decade a significant progress in the always searched objective of reducing the semantic gap between the mental model of software architect and the models handled by software tools (diagram, program etc.). This progress was materialized by the emergence of software architecture as an autonomous field of research/development in software engineering. The purpose of software architecture is to provide the concepts, mechanisms and tools needed to deal directly with various aspects of mental models related to a software system to realize

A mental model was always expressed in an informal box and line diagram. Each box deals with a precise functionality of a system. The lines correspond to interactions semantics or data flow. This view of software is often called software architecture and often represents the first step leading to the realization of a software product.

Software Architecture aims to easily and efficiently accommodate the mental models of architect and tries to join other engineering area, like computer architecture, by adopting the strategy of designing a system by assembling component.

To reach this objective, the procedural paradigm of interaction which prevails in the object model and the component models like JavaBeans, ActiveX, and EJB must be completely abandoned as was abandoned structured programming with the emergence of the object model[1].

A number of research works were conducted during the last decade concerning software architecture specification. These efforts resulted in the proposal of a great number of ADL. The work presented in [2] summarizes the characteristics of these ADL and discusses the main concepts of Software Architecture such as components, ports, composite component or configuration and connectors. Recently, UML 2.0, in an attempt to fill the gaps of UML 1.4, has introduced some mechanism in order to support the software architecture concepts [3]. Until now, proposed ADL have not known the awaited success. This is due to several factors, such as

- The orientations to solve problems in a specific domain[4]
- The orientation to deal with a particular architectural style [5]
- The exclusive use of formal languages like CSP[6] which are not suited for practitioners.
- The difficulty to design GUI based application.
- The supported component model deals usually with coarse grained application
- The component interaction model is usually based on the interface concept which heavily constrains the specification of architecture to a restricted set of well known topologies

fully influenced by the software mechanism, mainly the procedure call mechanism.

The IASA approach [7],[8] was introduced to fill most of these deficiencies. It offers an attractive alternative for the practitioners allowing them to specify architecture with a high degree of freedom from any software mechanism constrain. In addition, IASA support natively Aspect Oriented Software Architecture (AOSA) specification which reinforces one step further the modularity of a software system.

The IASA approach was validated by the effective realization of complex software systems mainly in the E-government and Telecommunication fields. In this paper we deal with the design of an EGovernment oriented application. Java web technology (ordinary Java classes, Java Beans, Java Servlets and Java Server Page) represents in this experience the targeted implementation technology.

In the remaining of this paper, we will briefly present, in section 2, the fundamental concepts of Aspect Oriented Programming (AOP) and AOSA. Section 3 deals with IASA fundamental model element and with *joinpoint* specification technique. A *joinpoint* is one of the basic elements of AOP. In section 4 we introduce the global objectives of the EGovernment project we realized using the IASA elaboration process. This later is presented in section 5 and section 6 partially shows its application in order to produce the EGovernment software product. Section 7 briefly presents the transformation technique used to generate the application in the targeted implementation technology and section 8 concludes this work by outlining some challenges facing the IASA approach in the design of Multi tiered application based on HTTP servers.

2 ASPECT ORIENTED PROGRAMMING

Aspect-Oriented Programming (AOP) is a recent software programming paradigm that aims at providing a better separation of concerns and reinforces one step further the modularity of a software system specification. Aspect Oriented Software Architecture (AOSA) is a recent trend in Software Architecture [9],[10]. Well known ADLs such as UNICON, RAPIDE, DARWIN, WRIGHT, ACME and *ArchJava* do not provide explicit support for AOSA. The most interesting works in Component Based Software Architecture deals with aspect either at a level of abstraction directly related to implementation level [11],[12],[13], or use an existing component model [14],[15] which is usually extended by the definition of specific interfaces, connectors and components. This is not the case with the IASA approach which supports natively AOSA at a high level of abstraction completely independent from any software mechanism.

AOP and AOSA are based on the following five concepts[16]: *joinpoint*, *pointcut*, *advice*, *weaver* and *advice insertion mode*. An *advice* represents the logic of a specific concern. The *joinpoint* indicates the location in the core business concern where the code must be altered by injecting the *advice* to produce the final system. The injection is achieved through a special mechanism called the *weaver*. The *pointcut*, specified usually as a regular expression, is a set of *joinpoint* where the advice has to be weaved. The *advice insertion mode* specifies how to operate the advice at a *pointcut* level. The most cited advice insertion modes are: *before (*the advice is performed before the *joinpoint*), *after* (the advice is performed after the *joinpoint*), and *around*.

The advice code corresponding to the *around* insertion mode, contains a first part that must be executed before the *joinpoint* and second part that must be executed after the *joinpoint*. The execution of a service with an *around* insertion mode is usually achieved as follows:

- The advice *before part* is executed.
- An optional call to a special instruction usually named *proceed* is made. This later launch the execution of the service attached to the *joinpoint* (a piece of code in programming language such as *AspectJ*[16].
- The advice *after part* is executed.
- The program execution is resumed just after the *joinpoint*.

The call to proceed may depend on the result of the advice *before part* logic. If proceed is not called, the *joinpoint* service is not executed, and the program execution resumes just after the *joinpoint.*

3 THE IASA BASIC MODEL ELEMENTS

In the process of defining the architecture of an application the following concepts are used in IASA: *access point*, *port*, *component*, *envelope*, *connector* and *action* [8]. These elements represent the fundamental concepts of the IASA ADL called SEAL (Simple and Extensible Action, Architecture and Aspect Language). The action concept, largely inspired from the OMG *Precise Action Semantic*|17, 20]), is used to describe miscellaneous architecture behaviors such as component interactions, port behaviors and component behaviors. In the following we first introduce the component model and the envelope concept then we discuss the concept of access point, ports and connectors.

3.1 The IASA component Model

The IASA component model defines a specific organization either for the external view applicable to any component (primitive, composite, COTS, legacy code) or for the internal view |8]. The external view is represented by the concept of envelope. The internal view consists of two parts: the *operative part* and the *control part*. The *operative part*, which appears at the top of the IASA component graphics

notation, contains the components achieving the objectives of the core business aspect. Any component which has an internal structure different from the IASA internal organization of component is said a primitive component. COTS, legacy code and component written in a programming language are examples of primitive components.

The control part which appears at the bottom of the IASA component graphics notation, is composed of a controller, which is a specific component dedicated to control the operative part and a number of components handling the technical aspects (i.e. tracing, exception, transaction). The controller is a mandatory component of the control part. The components handling technical aspects are usually called aspect components.

3.1.1 The Envelope Concept

The main goal of the envelope is to provide a total isolation of the internal view of a component from the external world. The envelope is mandatory in the process of instantiating a component type. In IASA it is not possible to instantiate a component type without specifying an envelope. For a given component type, it is possible to associate a number of different envelopes, each one is used in a specific situation. The envelope represents a sort of clothes an instance of a component type wears in a specific situation. Hence, it is possible to associate instances of the same component type with different envelopes either at the same level or at a different level of the composition hierarchy describing a composite. The envelope specifies for a component instance its deployment case which describes the deployment environment (machine, operating system, process, application server) and the exact nature of the component in such environment (PROCESS, MAIN THREAD, THREADS, SERVLET, EJB etc.).

An envelope hosts all the resources needed to support communication aspect (i.e. adapters), to enable the specification of connections involving the port's structural elements and to handle aspects weaving operations of code. In addition, the envelope concept enables to integrate non IASA components in a design (i.e. legacy code, COTS, etc.) and to inject advices at their port level.

3.1.2 Aspect Components

An aspect component is the central place where the aspect advices are specified[18]. It shares the same component model with business component. An aspect component is instantiated only in the control part. Only one instance of an aspect component may exist in an entire application. The aspect components are oriented to support technical aspects which usually correspond to non functional properties of the system. In IASA, a technical aspect is identified by a unique aspect identifier (*AspectId*).

An aspect component, as defined in the current implementation of IASA, may have three kinds of aspect ports: *advice ports*, *advice inhibitor ports* and *interest for an aspect port*. The *advice port* gives access to the advice logic provided by the aspect component. The *advice inhibitor port* is used to specify that the architect explicitly ignores an *interest for an aspect* in its design. The *interest for an aspect* is usually shown in the external view of a business component. The main role of an *interest for an aspect port* is to relay, according to the provided advice logic, the management of an aspect to the external world, usually represented by a more complex component.

3.1.3 IASA Link Component

The link component (*LinkCmp*) is used to represent the same component instance across the composition hierarchy of an application, in order to produce lucid and clear architecture specification and to avoid proliferation of delegation connector. The *LinkCmp i*s widely used in GUI design based on IASA visual component [7]. It is also used to represent the execution environment and external component to an application (i.e the file system, a DBMS, an HTTP server etc..). The *LinkCmp* provides more than the concept of shared component of FRACTAL[19]. The *LinkCmp* provides means to attach to a same component instance, different personalized external view, in the same or in different level of the composition hierarchy. The personalization of *LinkCmp* is mainly achieved by using the *alias* construct of the SEAL language either to personalize the action name describing the port behavior [20] or to associate an action to a specific *aspect insertion mode* (*before*, *after*, *around*)[18]. Hence, in each *LinkCmp*, the same action may have a different name and the same *aspect insertion mode* may be associated with different actions

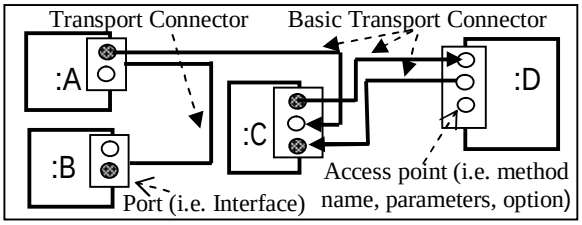

Figure 1: IASA Connectors based on port's element

3.2 The Access Point Concept

An *access point* is the smallest structural element in the specification of an application [8]. It is used to define the ports of components. An access point exposes required or provided resources which may be data or operations. Communication mode and the resource time validity are among the properties of an access point. An access point may

be wired in an independent manner to another access point which is hosted in the same or a different port.

In current software architecture models and tools, an interaction point, usually represented by an interface is considered as an atomic element despite its complexity**.** It is not possible to deal separately with elements defining the structure of such interaction points (i.e. method, method parameters). A connection's endpoint, usually named *role*, is connected only to an interface. In current software architecture models, it is not possible to define a connection between port's elements as in Figure 1.

In order to allow more accurate and practical specification of architecture, IASA defines specific access points according to their global roles in a component. The specific access points are organized into two categories: The Data Oriented Access Point (DOAP) and the Action Oriented Access Point (ACTOAP). Figure 2 shows the IASA graphics notation for access points.

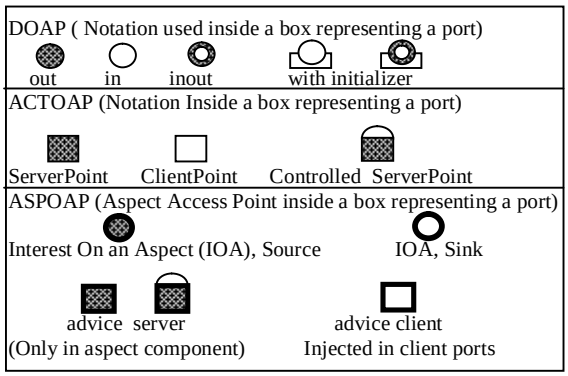

Figure 2: IASA graphic notation of Access Point

A DOAP is used to transfer data of any type. It is provided with an attribute specifying the data direction (*in, out,* and *inout*). The definition of a new specific DOAP is achieved by providing its implementation level representation which follows a specific naming and definition style. The naming style uses the data type name followed by *DataPoint (i.e. IntDataPoint).* The definition style is based on the name of the supported data type and a template file written according to the targeted implementation technology. The sample *ArchJava* code in Figure 3 shows how a template file is used to build an access point oriented to send or receive Java *String* type.

An ACTOAP is provided with a set of actions supported by the service (*actionSet*). Regarding the associated service, an access point plays one of two basic roles: a *server* or a *client*. The server role is played by the *ServerPoint* and the client role is played by the *ClientPoint*

The *ServerPoint* manages a second set of actions called the *refinedActionSet*. Each element of the refined action set is associated with only one action in the *actionSet*. A *refinedActionSet* describes one step further the refinement process of the associated actions.

```
package iasa.datapoints; // StringDataPoint definition
public class StringDataPoint extends DataPoint{
   private String data;

  StringDataPoint(StringDataPoint sdp, int dir) {
     copy(sdp);
     this.dir = dir;
  }
   StringDataPoint(String s, int dir) {
     data = new String(s);
     this.dir = dir;
     this.timeValidity = 0;
  }
  String get() throws   InvalidAccessToInDataPoint,
                        AccessPointTimeOut {
       getValidate();
       return new String(data);
  }
  void set(String s) throws  InvalidAccessToInDataPoint,
                             AccessPointTimeOut {
            setValidate();  data = new String(s);}
  public  void copy(DataPoint dp){
     data = new String(((StringDataPoint)dp).data);
  }
  public  void startTimer(){} // Not yet Implemented
}
```

Figure 3: *String* DOAP in ArchJava

3.3 Ports

A port is a technique for grouping related access points. It maintains an abstract and a concrete views. The concrete view may be any model, provided with a clear way leading to the implementation level (i.e. an interface based port, a UML port, an *ArchJava* port). The abstract view is represented by three elements: the concept of access point, the actions associated with access point and a behavior. The port's behavior is represented by a set of valid rules defined in the SEAL language. Each rule shows how the required or provided resource must be used. When connecting two ports, the connector is said to be valid if the supported interactions use compatible port's behaviors. Figure 4 shows a SEAL partial description of port types used in the external view of the *CivilStateCmp* component type shown in Figure 12.

For an efficient and clear specification of connections between components, a number of ports organized in four categories are predefined in IASA: *regular ports, aspect ports, controlled ports*, and *standard ports.* Figure 5 shows the main graphic notations of ports used in IASA.

A regular port is basically composed of a fixed number of access points. In the current version of IASA, three regular ports are predefined: *ClientPort, ServerPort* and *DataPort*. A *ClientPort* contains one *ClientPoint* and zero or more *DataPoint*. A *ServerPort*, as a consequence of the previous definition, contains one *ServerPoint* and a number of *DOAP*

```
// SEAL ADL: file :CivilStatePortType.seal
package eapc.ports;
import eapc.doap.*;
// Action context definition
actioncontext  citizen_basic_actions {
                // all actions of this context  are abstract
                actions  birth , death,  marriage,
                          divorce, family;
              }
 }
port  { // Port type definition
     port  DocumentPort {
       accesspoint{
        ServerPoint  docSp (0, SYNC);
        CitizenIdDataPoint   citizenId (OUT, 0, SYNC)
       }
       actioncontext  {
                uses citizen_basic_actions;
       }
       behavior {
         rules birth_r, death_r,  marriage_r,
                          divorce_r, family_r;
         rule birth_r {
                          precondition:;
                          pattern: birth;success;
                          postcondition:;
                          fail:;
         }
         rule death_r {  // empty section may be omitted
               pattern: death;success;
         }
         rule marriage_r {..}
         rule family_r {..}
         rule divorce_r {..}
       }// end of port type behavior
  } // end of DocumentPort type definition
  // other port type definition
}// end of global port type definition
```

Figure 4: Port Type specification with SEAL

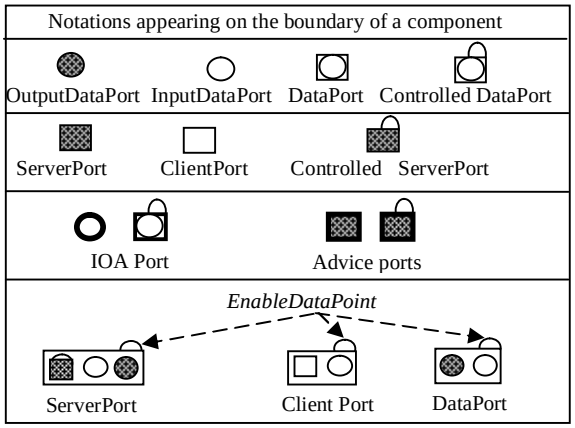

Figure 5: IASA graphic notation of Ports

An aspect port is either an *advice port* used by aspect component to provide advices or an *interest for an aspect port* used by business component to express an interest for one or more aspects. An advice port is a *ServerPort* provided with actions explicitly associated with supported aspect activation mode. For now, the supported advice activation mode are: *aroundFirstAction*, *AroundLastAction*, *proceeedAction, beforeAction* and *afterAction*.

A controlled port (figure 5) is a port provided with the *EnableDataPoint* DOAP, a *ServerPort* provided with the *ControlledServerPoint* instead of a *ServerPoint*, or a port provided with both control techniques. Standard ports are oriented to support well known connectors such as a standard protocol or the interaction with a standard execution environment such as an operating system or an application server.

3.4 IASA Connectors

The IASA connector model is largely inspired from computer network architecture. The model provides a behavioral view and a structural view. The behavioral view describes an interaction and the structural view defines the infrastructure needed to transport the interaction. The connector infrastructure is based on two kind of fundamental connector elements:

- *Transport Connectors* which are point to point connectors composed by *Basic Transport Connector*, which can connect only two compatible access points (Figure 1).
- *Service connectors* which are primitive component oriented to support specific interconnection functionality (distribution, multiplexing, load charge balancing, resource location etc.) as described in [21] and [22].

While defining the architecture of an application, the designer focuses all his energy on choosing the accurate component, specifying the correct interaction and defining the interconnection infrastructure. Hence, the designer is not concerned by the definition of new *Service Connectors* or *Transport connectors* which are predefined in the IASA approach and have a complete realization in the supported implementation technologies. The designer is a connectors user and not as a connector designer. The definition of the interconnection infrastructure is achieved in IASA by cascading *Service Connectors* using *Transport Connectors.*

3.5 Pointcut specification

A *pointcut* is the set of *joinpoint* where an advice is weaved. In IASA, a *joinpoint* is localized only at port level [18]. It may be any action attached to an ACTOAP, the implicit actions of sending and receiving data on a DOAP or any rules defining the port's behavior.

A *joinpoint* is identified by a hierarchical name specifying its location in a design. A complete *joinpoint* name (or absolute name) is composed of four parts separated by the dot symbol (i.e. docRep.spBirth.sap.birth). The first part is either a component type or a component instance used in the design. The second part is either a port type or a port instance. The third part is either an access point or a rule name and the last part is usually an action name. When the third part is a rule name, the last par may

be the **rule** keyword. The **rule** keyword specifies that the *joinpoint* is any valid trace of the specified rule. The component (instance name or type name) represents the highest level (left part) of the *joinpoint* identification. The lowest part (right part) of the *joinpoint* identification is either an action name or the *rule* keyword.

The aspect identifier (*AspectId*), the star character symbols, and SEAL keywords (i.e. (*serverport, clientport, dataport, interest, rule, send, receive*) may be used to specify *joinpoints* generic name in the process of a *pointcut* definition. Operation on set (i.e. union, difference) may also be used to define new pointcuts from others.

4 THE APPLICATION AND ITS DOMAIN

In this experimental study, we realized for the Center for the Development of Advanced Technologies a software system which enables the citizen to access through the internet to various services of a local government institution called APC (the town council). The most required services from the APC are the production of official documents exposing important events such as the birth certificate and the marriage certificate. Inside the APC, the service delivering such official documents is called the *Civil State Service*

Currently a citizen requiring any of these certificates must present himself to the APC with necessary proof documents and ask an APC agent to deliver him the desired documents. In addition to consume citizen energy and money, the most important drawbacks of the current situation are the long time passed waiting the production of a document (in some situation a day represents the time unit) and the high rate of errors produced in the delivered document since this operation is achieved manually.

The major goal of the targeted EGovernment application is to reduce the impact of the cited drawbacks, by enabling citizen to participate through the internet in the document production process. The other major goal is the preparation of the APC and other local government institutions to move to the intensive use of internet technologies by setting a whole EGovernment system providing all government services required by citizens.

The EGovernment system for the APC must provide efficient solutions to the following challenges:

- A huge amount of data describing citizen events has to be captured.
- A high degree of security must be guaranteed for accessing critical part of the system and personal data

The first challenge was solved by defining a strategy where the citizen is indirectly involved in the process of entering citizen data. The main benefit of this strategy is the fact that a citizen natively makes a best effort to guarantee the correctness of data describing him or any of his relatives. In addition, with this strategy, the citizen participates efficiently to highly reduce the problem of errors produced when delivering documents. This first challenge was solved as a part of the core business aspect of the system.

The second challenge was solved by the use of a predefined aspect component belonging to the security aspect of IASA. This aspect component provides many security facilities such as user management, user authentication, access right management, session management and solutions to well known security problem (SQL Injection, Cookie poisoning, session hijacking etc.).

5 THE IASA ELABORATION PROCESS

The IASA elaboration phase in the design process of a software system is completely automated in the context of IASA STUDIO (Figure 6). This process is obviously based on a previous analysis phase of the problem. The most interesting result from the analysis phase is the emphasizing of all needed external components which interact with the application being designed. The elaboration process follows a recursive top down strategy made of a two great phases: An initialization phase and a recursive phase.

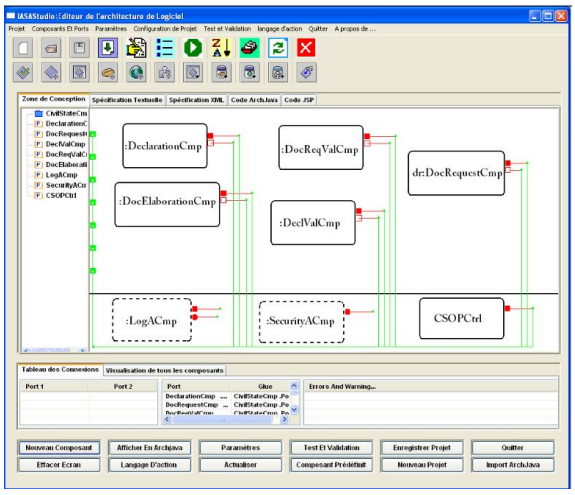

Figure 6: *CivilStateCmp* in IASASTUDIO

The initialization phase is concerned by definition of the external view of the whole application and the determination of external component. The recursive phase deals with the design of component's internal view and its first step target the whole application internal view. In the next section we present a partial view of this process. A complete description may be found in [7]

5.1 Phase 1: Project Initialization phase

5.1.1 Step 1

Based on a previous analysis step of the problem, this step starts with the specification of the application name (AN) and the deployment architecture (DA). Regarding IASA, the application itself is considered as a component type which may be instantiated in more complex application.

5.1.2 Step 2: External view definition

The goal of this step is to define the provided and required services, the definition of actions, the organization of service inside ports and the specification of port's behavior. The following tasks are performed in an iterative way.

Global informal description: This phase begin with an abstract definition of the system or component, in the form of only one box from where leave and arrive several arrows. Arrows represent the provided and needed services, data or controls. To eliminate any ambiguities, the boxes and arrow specification must be accompanied by a narration explaining the total functionalities of the system and the semantic associated with the arrows. The narration may also contain the requirements and constraints fixed by the customer.

Resource organization: This task represents the first task towards formalizing the external view. From the preceding definition, we must define the provided and required resources (services and data). The definition of services is accomplished by specifying action names and optionally action input and output pins.

Definition of the external ports: The required and provided resources are gathered in ports according to the supported port type (regular port, controlled port etc.). The behavior of a port is then defined based on the actions defined previously.

At the end of this first phase, a full SEAL description of the external view is produced by IASA STUDIO and ready to be transformed in a supported implementation technology such as ArchJava[23].

5.1.3 Step3

Let *LCmp* a set of component and deployment architecture pair initialized as follows:

LCmp = {(AN, DA)}

5.2 Phase 2

This is the recursive phase of the elaboration process. For each pair (ANX, DAX) from *LCmp* , the following steps have to be performed.

5.2.1 Step 1: Internal view elaboration

This step is concerned by the elaboration of ANX internal view and the definition of the controller behavior. This step is realized as follow:

Let *ILCmp* an intermediate set of components and deployment architecture pair initialized to empty.

While the internal view of ANX is not stable (the stability analysis is done using the SEAL interpreter) perform the following design action

- Find the component type needed to realize ANX. Here it is recommended for the first phase to create new component type instead of modifying existing instantiated component type. Modification of instantiated type may create design exceptions which force the designer to restart from a stable point in the design process.
- Define the external view for new component type (Port and action)
- Establish / modify / Remove connectors
- Define connectors SEAL actions needed for the interaction definition
- Define a mapping between external view and internal view: This mapping shows exactly which component handles a provided resource and which one need a required resource.
- Défine / Adjust the behavior of the controller.
- Verify the component's stability using the SEAL interpreter
- Adjust the external view of new component type
- Specify the deployment case for the component instance according to initial deployment architecture DAX
- Add new introduced component type to *ILCmp* set.

5.2.2 Step 2

End of internal view elaboration of ANX. The ANX component is stable

5.2.3 Step 3

Prepare the next recursion. Add new component types to LCmp:

LCmp = LCmp + ILCmp.

5.2.4 Step 4

End of the design elaboration process.

6 THE APPLICATION DESIGN

In the following we partially show the use of the just introduced elaboration process in the design of the previously described application.

6.1 Phase 1: Project Initialization phase

6.1.1 Identification and deployment architecture

The targeted application is named E_APC. The deployment architecture is an ordinary three tiers architecture based on an HTTP server provided with a Java Servlet Engine such as Apache Tomcat. The other important element of this architecture is a database management server. We used for this first implementation the MySQl server.

6.1.2 The external view

Global description of the system: Figure 7 presents a global view of the system. It is clear that such a view is ambiguous. It requires a complementary narration and could not be treated by software tools. Such view is usually elaborated in the analysis phase. Here it serves as a base for the first formal specification using IASA notation (Figure 8). Usually the arrows correspond to services and may include many actions. Hence, the *Document* arrow includes at least all the actions performing the production of specific documents such as birth certificate, family certificate etc. The *Opinion Poll* arrow includes actions of voting and action for displaying current vote score.

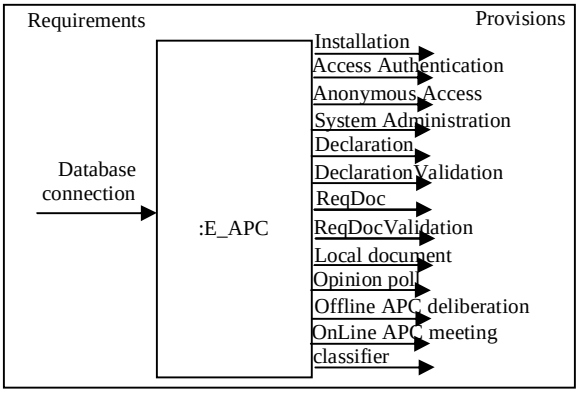

Fig 7: Informal specification of E_APC

Resources organization in the external view: The preceding informal view is transformed into a more precise view (Figure 8) which would be the starting point for a successive operation of refinement until reaching the desired software product. Figure 8 shows all the ports of the system using the IASA notation and figure 9 present a partial view of the E_APC in the SEAL language. Within each port are defined a number of actions dealing with objectives assigned to the port.

The interesting observation on the formal specification of the E-APC, according to IASA approach notation (Figure 8), is the lack of port dealing with access control. This situation is in fact due to the support of aspect oriented software architecture in the IASA approach. While using the aspect paradigm in system design, we usually use following rule:

- Establish the separation of concern between the core business aspect and the technical aspects.
- Elaborate separately each aspect.

Technical aspects have not to be considered when designing the core business aspect. This later has to be designed for an ideal environment which provides necessary technical support to safely operate the core business aspect and decides where and when to place the support. Components realizing the core business aspect fully implement the concept of obliviousness [24] since they are completely unaware of the existence of the technical aspect

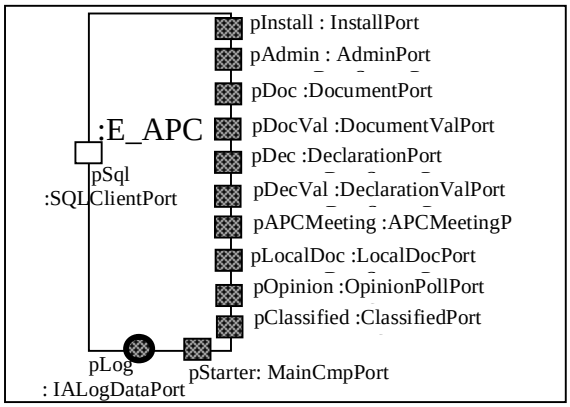

Figure 8: E_APC external view using

```
// SEAL ADL: file : E_APC.seal
package eapc.component;
import IASA.aspect.*;
import eapc.ports;
component E_APC{
   ports { // The external view
     // required Services
          SQLClientPort        pSql;

     // Provided services
          MainCmpPort          pStarter;
          InstallPort          pInstall ;
          AdminPort            pAdmin ;
          // … other regular ports here
          ClassifiedPort       pClassified;

     // Interest for an Aspect (IA) DOAP
          IALogDataPort        pLog;
   }
   operativepart {
     components { …….. } connectors {  …….. }
   }
   controlpart  {…… }
 }
} //End Description of E_APC component
```

Figure 9: Partial SEAL description of *E_APC*

With Aspect Oriented Design, the application developer does not need to worry about the nonfunctional services. It is the aspect developer who, in addition to designing and writing the code of the service itself, manages the integration of that service into the application. The advantage is that the specialized aspect developer has a better understanding of the service than the application developer, who is only a user of this service

6.2 Phase 2: internal view *LCmp* elements

In the first step of this recursive phase, the *LCmp* set contains only the *E_APC* component which is the targeted application. Figure 10 shows some fundamental components of the *E_APC* internal view. The operative part is composed of six business components and the main view of the application IHM. The control part contains two

aspect components in addition to the mandatory controller component (*EAPCOPCtrl*).

6.2.1 Managing the security aspect

The operative part is designed without taking any care concerning security and logging aspects. If at this stage any service has to be secured, the only thing designer has to do is to connect the advice port of the security component to the port providing the service to secure. This operation, called *aspect injection*, is achieved by specifying the *pointcut* containing the actions concerned by the security advice. Figure 11 show how the security aspect is injected in all *ServerPort* of all business component except those component which do not need to be secured.

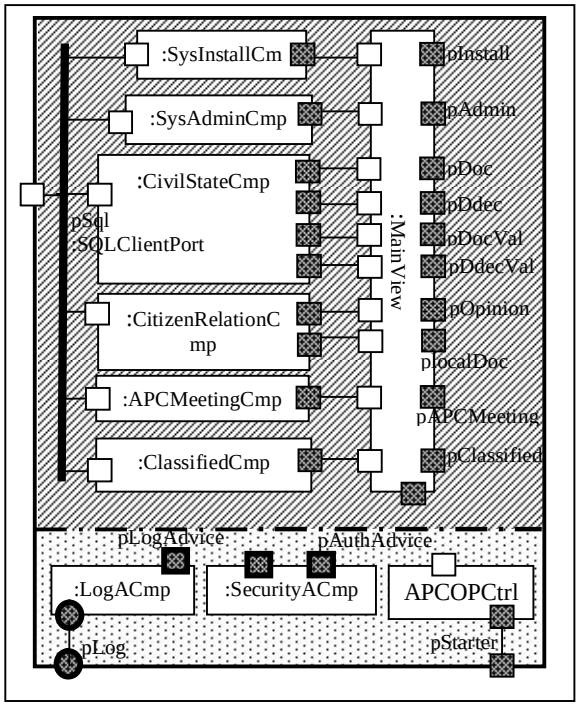

Figure 10: partial Internal View of *E_APC*

This step of phase 2 is completed with the introduction of six new component types: *SysInstallCmp, SysAdminCmp, CivilStateCmp, CitizenRelationCmp, APCMeetingCmp, ClassifiedCmp.* The used aspect components (*LogACmp* and *SecurityACmp)* are not concerned by further design step since they are completely defined and provided by the IASA Design Environment. The completion of this step results also in the complete definition of the external view of new instantiated components. Hence, before joining the *LCmp* set, a component type must have a well defined external view. Further design steps are usually concerned with the internal view design of components from the *LCmp* set.

This step is repeated for each new introduced component type. In the following we will focus our interest in the design of the *CivilStateCmp*, which is the most important component in the E_APC application.

```
// SEAL ADL: file : E_APC.seal
package eapc.components;
import IASA.aspect.*;
import IASA.ports.*;
component E_APC{
    ports { .....}
    operativepart {
      components { …….. }
       connectors {  …….. }
   }
controlpart {
    components {
       APCOPCtrl apcOPController;
       SecurityACmp secCmp;
       LogACmp logCmp;
    }
   aspect { // Aspect pointcut and advices management
     pointcuts {
       all_services={ serverport}
       not_secured = { CitizenRelationCmp, APCMeetingCmp,
                       ClassifiedCmp};
       partial_secure = all_services – not_secured
     }
     advices {// Advices Management
        inject secCmp. p.AuthAdvice  around  partial_secure;
       }
      interest {// Interest for Aspect Management
      }
  }
} //End Description of E_APC  component
```

Figure 11: security *pointcut* definition and injection

6.2.2 The internal view of *CivilStateCmp*

It is composed of a number of components, each one oriented to handle a specific functionality of the civil state department of the APC (figure 12).

The components *DeclarationCmp* and *DocReqCmp* are oriented to enable the participation of the citizen in the process of populating the E_APC databases with accurate data concerning them. *DeclarationCmp* handles the declaration of new events such as birth, death, marriage or divorce. *DocReqCmp* is used to request miscellaneous certificates and, in the same time, is used to enter citizen data if these later were not yet captured in a previous declaration or document request.

The validation components (*DeclValCmp* and *DocReqValCmp*) are used to validate data entered by citizen either with *DeclarationCmp* or *DocReqCmp*. The *DocReportCmp* is used to produce the desired certificate.

We notice in the internal view the use of link component to represent the security and log aspect component previously instantiated in the E_APC internal view. With this technique the designer can manage the injection of the same aspect at various level of the design hierarchy.

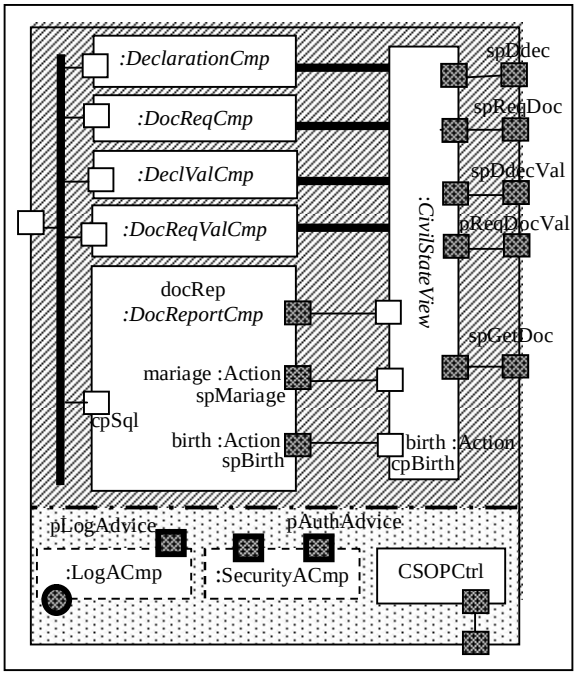

Figure 12 : *CivilStateCmp* partial internal view

```
// SEAL ADL: file :CivilStateCmp.seal
package eapc.component;
import IASA.aspect.*;
import eapc.ports.*;
component CivilStateCmp {
    ports { .....}
   operativepart {
      components { ……..  } connectors {  ……... }
   }
  controlpart {
    components {
       CSOPCtrl csOPController;
       LinkCmp (E_APC.secCmp ) secCmp ;
       LinkCmp (E_APC.logCmp ) logCmp;
      // Since an aspect component is a singleton, the two
      // previous line may be written in a more clearer way
      // LinkCmp (SecurityACmp) secCmp ;
      // LinkCmp (LogACmp ) logCmp;
    }
   aspect { // Aspect pointcut and advices management
     pointcuts {
       not_secured = { DeclarationCmp, DocRequestCmp };
       // other pointcut definition …….
     }
     advices {// Advices management // Adjust security
injection
         remove secCmp. pAuthAdvice  from not_secured;
              // other advices management  …
      }
     interest {// Interest for Aspect management
     }
  }
} //End Description of CivilStateCmp component
```

Figure 13: Adjusting security aspect injection

In the case of the internal view *of CivilStateCmp* we notice that there is no need to secure the access for *DeclarationCmp* and *DocReqCmp* since these components are oriented to encourage citizen to enter their data and participate in the whole process of capturing citizen miscellaneous information. However, in the previous step we have injected the security aspect at all server port, which means that all actions at those ports are targeted by the security advice. Consequently, the server ports (*spBirth, spMariage* etc..) of *DeclarationCmp* and *DocReqCmp* find themselves secured. To solve this sort of problem we have two solutions:

- Review the injection statement in the internal view of E_APC. With this solution we must restart the design from previous step and the top down methodology is then noised by this kind of design decision
- Avoid the design noise in the top down design strategy by adjusting the security aspect injection of a previous step. In our case adjusting operation is realized using the aspect *remove* capability of the SEAL language (Figure 13). This tactic seems to be the best suited in most top down design process since it allows adjusting efficiently previous design decision without the need to return to a previous completed design step.

7 GENERATING THE APPLICATION

7.1 Fixing the deployment properties

Until now, the design of the E_APC appears to be similar to an ordinary application, since no element in the design indicates that the application is a multi-tiered one based on an HTTP server provided with a Java Servlet Engine. Actually this is one of the main objectives of the IASA approach: The independence of an IASA specification from the software mechanism and the deployment architecture. Hence a same component may be deployed as a Java Servlet, an EJB, an ordinary application or a VLSI chip. In practice, the final form of generated code of an application depends on the application's *deployment map* which in turn is strongly impacted by used interaction technology used (i.e. standard protocol such as FTP, HTTP, SOAP, RMI etc..) and library components.

The *deployment map* (Figure 13) is the specification of the *deployment case* for each component instance of the tree representing the composition hierarchy of an application. The *deployment case* is the form a component may get in the chosen implementation technology (PROCESS, THREAD, MAIN_THREAD, SERVLET, APPLET, EJB etc.). Since the E_APC is hosted by a web server provided with the Servlet Engine, the deployment cases are restricted for the whole application to APPLET, JAVASCRIPT, JSP, SERVLET, BEAN and CLASS

The code generation of an application is guided by the *deployment map* and uses a set of transformation rules. These later take a SEAL

description and produce a concrete description in the desired implementation technology. Regarding the OMG MDA (Model Driven Architecture) [25], IASA architecture without a *deployment map* corresponds to the MDA Platform Independent Model (PIM). The PIM is transformed on a Platform Specific Model (PSM) once a *deployment map* is applied, or defined, for the IASA specification.

7.2 The main transformation rules

In IASA the transformation rules mainly target the generation of the envelope. The transformation of behavior written in SEAL (mainly the behavior associated with the controller) is usually straightforward, since an action usually corresponds to a service call (i.e. procedure call, protocol action such as HTTP methods). The main goal of the transformation rules is to produce code in the chosen implementation technology, needed to support:

- The specification of various kind of interconnection based on ports (*Transport Connectors*) or on access points (*Basic Transport Connector*)
- The management of aspect injection (weaving and ordering injected aspects)

In the following we briefly present the main transformation rules used in the process of generating the E_APC application.

7.2.1 Generating the application without aspect

The following rules are used to generate a pure business application, without any technical concern.

- A Business component may be a JSP page, a Servlet, an ordinary Java class, a Java Bean, an Applet or a JavaScript.
- An aspect component is always deployed as an ordinary java class (or a java bean) provided with a number of static methods. Each static method is designed to handle a specific action belonging to the supported *aspect activation mode*.
- Since the application belongs to the EGovernment domain, any browser communication bypassing the HTTP protocol is not supported (i.e. Java RMI). Communications using protocol built on top of HTTP (i.e. SOAP) are fully recognized. As an example, it is not possible to deploy a component as an applet if this component is provided with port using a non HTTP communication protocol.
- For component deployed as JSP, Applet or JavaScript, there is no need for connector adapter since all used connectors and ports are based on standard protocol (HTTP, SQL) completely defined in IASA. Component deployed as Ordinary classes, Java Bean and Servlet may use other connectors (RMI, CORBA, IP Socket etc..) requiring adapters on connected ports.
- Due to the stateless nature of the HTTP protocol (no connection information maintained between transactions), the *after activation mode* is always inhibited for advices injected on ports of components deployed as JSP or Servlet. In the same context, the *aroundLastAction* of the *around activation mode* is only executed if the advised *joinpoint* is not executed.

```
////// File E_APC.dpy
// Description of recognized deployment architecture and
// deployment case
package eapc.component
component E_APC {
  architecture {
     environment tomcat {
        machine  localhost;
        container tomcat5.5 ; //apache Tomcat 5.5
        namespace  eapc ;  //localhost:8080/eapc
        os UNIX; // Generic name used.
        deploymentcase {APPLET, JAVASCRIPT,JSP,
                        SERVLET, BEAN, CLASS}
     } // Many environment may be defined.
     environment J2ee {
          machine  eapc.cdta.dz;
          container JONAS ; //apache Tomcat 5.5
          namespace  eapc ;  //localhost:8080/eapc
          os UNIX; // Generic name used.
          deploymentcase { APPLET,
JAVASCRIPT,EJB,
                        JSP, SERVLET, BEAN, CLASS}
     }
  }
// Definition of the deployment map.
// Many maps may be defined for the same application
deploymentmap map_for_tomcat {
       // go across the composition tree  and associate for
each
       // component instance a supported deployment case
       // regular expression may be used
        deploy this as JSP in tomcat; // produce E_APC.jsp
       // in case where the deployment case is the same,
        // use the keyword all or rall. Extension .jsp
       // is appended to Absolute component instance name
```

Figure 14: Deployment specification

7.2.2 Generating the application with aspect

The injection of aspect is supported by additional transformation rules fully compatible with the just introduced ones. The weaving of injected aspect is achieved at the envelope level. With this technique we guarantee a complete isolation of component type from the instantiating environment. Hence, there is no need at all to touch any realized code (class, servlet, bean etc..) representing IASA components.

The aspect injection process results in the modification of the port behavior. This modification targets either an action or a complete port behavior rule. Due to the application nature, we notice the simple structure of port behavior rule which are often reduced to a simple action (Figure 4). This was not

the case with another experience [26], where some ports were provided with complex behavior.

The examples in figures 4,11,12,13, and the following figures briefly outlines some elements of the transformation process. Figure 4 shows the original port's behavior designed far from technical aspect. This port's behavior is attached to the server port (*spBirth*) of *DocReportCmp* and the corresponding client port (*cpBirth*) of *CivilStateView* (Figure 12). Usually we encounter compatible behavior in the connected client and server ports.

Figure 15 shows the behavior of the client port after the injection of the security aspect as specified in figure 11 and 13. According to IASA policy, the weaving of aspects is located in the client port connected to the targeted server port containing the *joinpoint*. The code in figure 15 is an intermediate code produced by the SEAL interpreter after executing the aspect management operation (inject, remove, inhibit etc..) and before the generation of the code in the targeted implementation technology.

```
rule birth_r {
  precondition:;
  pattern: ArounfFirstAction; proceedAction |birth;
       aroundLastAction;success;
  postcondition:;
  fail:;
}
```

Figure 15: weaving aspect at port behavior level

```
<a href="ports/cpBirth.jsp"> birth certificate </a>
```

Figure 16: inside the *CivilStateView* code

Figure 16 illustrate the technique used in IASA approach for a component to contact the external world. The service call represented by a link in the *CivilStateView* deployed as a JSP page, is first directed to the component's port (*cpBirth*) and not directly to a specific component. This technique insures the total independence of a component from the external. Hence the same component type may have distinct instances connected to distinct components through the same port.

```
<!-- Redirect to the envelope port -->
<jsp:forward page=""../envelope/active/cpBirth ">
```

Figure 17: Inside *cpBirth.jsp*

To insure the total isolation of a component type from any instantiating environment, the service call is redirected to the envelope used to instantiate the component type (Figure 17). Further operation on component will have no impact on the component instance. All ad hoc modification needed by the placement of adapter and injection of aspect are located in the envelope. Figure 18 shows the connector implementation which is represented by a redirection of the service call from the envelope to the envelope of connected server port (*spBirth).*

```
<%@ page contentType="text/html; charset=utf-8"
language="java" %>

<!-- Redirect to the connected port. use of absolute parth -->
<jsp:forward
  page="/CivilStateCmp/docRep/envelope/active/spBirth.jsp"
>
```

Figure 18: Inside the envelope port *cpBirth.jsp*

```
<%@ page contentType="text/html; charset=utf-8"
language="java" import="iasa.security.*" %>

<!— Boolean ASPOAP Transformation process -->
<%=boolean proceedState = true%>

<!-- ------ ----- Around body part 1 -->
<%  AuthACmp.pAuthAdvice.aroundFirstAction;
    proceedState =
        AuthACmp.pAuthAdvice.proceedAction;
    If (proceedState) {
%>
<!-- ------ ----- Around body end of part 1 -->
<!-- Redirect to the connected port. use of absolute parth -->
<jsp:forward
page="/CivilStateCmp/docRep/envelope/active/spBirth.jsp">

<!-- ------ ----- Around body part 2 -->
<%}
   AuthACmp.pAuthAdvice.aroundLastAction;
%>
<!-- ------ ----- Around body end of part 2 -->
```

Figure 19: The envelope after aspect Injection

Figure 19 presents the result of weaving the security aspect using the *around activation mode*. We notice in figure 19 that the after part of the *around activation mode* is executed only if the proceed part indicates that the join point is not executed.

8 CONCLUSION

The work described in this paper is an actual experience where a software architecture approach is used to realize a complex EGovernment software system deployed in Java web technology. In most works related to software architecture, the examples used to validate a model or to show its various qualities are usually reduced to very simple examples in the context of well known architectural styles (pipe and filters, client server, blackboard, layered model etc.). To our knowledge, there is no published work relating such a true experience where a software architecture specification is deployed in the context of a multi-tiered architecture based on an HTTP server provided with a servlet container.

This true experience has also shown how a software architecture approach could lead to the fast

realization of complex software system. In this experience, the IASA approach and its software elaboration process were conducted in parallel with the realization of the same product using the EJB component model and the elaboration phase of the Catalysis object oriented design process [27]. This experience showed the high flexibility and the power of IASA to easily handle software architecture specification and to reduce the realization time. Compared to the object oriented project realized using an object oriented approach based on EJB, the realization time in IASA using Java web technologies was by far the shortest.

This time performance may be explained by the following facts

- The use of EJB in the context of an object oriented approach requires the direct control of several technologies (Servlet, JSP, JavaBeans, EJB, XML etc.) and concepts, whereas in IASA, the only concepts to be acquired are the fundamental concepts of software architecture (component, port, connectors) and the various facilities to compose an architecture provided in IASA, mainly link component and access point. All complex concept of the chosen implementation technology are abstracted by the IASA approach.
- The efficient support of the concept of composition which allowed the designer to follow a top down design process. The composition concept is not natively supported by the EJB component model.
- The link component concept which allowed the designer to elaborate simple a clear design which are easy to maintain and evolve.
- The aspect management facilities provided by SEAL language. These concepts allowed the designers to follow a clear and organized top down design process as shown in this paper without the need to return to a previous step for adjusting previous design decision.

The aspect oriented approach in IASA played a fundamental role in reducing realization time. With AOSA, the application developer does not need to worry about the nonfunctional services and properties. The application developer focuses all his work in the core business aspect and do not consider the technical aspect. It is the aspect developer who, in addition to designing and writing the code of the aspect service itself, manages the integration of that service into the application. The advantage is that the specialized aspect developer has a better understanding of the service than the application developer, who is only a user of this service.

This achieved work has also shown some challenges the IASA approach is currently facing. An important research effort is needed to solve these challenges in future works. Currently, the main challenge is located in the transformation process from an abstract view described in SEAL to a concrete view, represented in one or more implementation technologies. The transformation process, as in [26],[28], produces a great amount of code. This situation is mainly due to the envelope concept which is associated with each component instance. As an example, for each component deployed as a JSP page, the envelope is represented by a number of JSP page equals to the number of port of that component. Optimizing the number of envelope in the transformation process represents one of the planned future works in the IASA approach

REFERENCES

[1] C. Carrez: Behavioral Contracts for Component' Ph.D. Thesis, ENST, Paris, 2003 (In French).

[2] N. Medvidovic, R. N. Taylor: A Classification and Comparison Framework for Software Architecture Description Languages, IEEE Transactions on Software Engineering, Vol. 26, no1, pp. 70-93, January 2000

[3] Unified Modeling Language: Infrastructure, version 2.0, 3rd revised submission to OMG RFP ad/00-09-01, OMG, January 2003

[4] S. Vestal: Scheduling and Communicating in MetaH. Real-Time Systems Symp., pp: 194-200, Raleigh-Durham (NC), 1993.

[5] N. Medvidovic, R.N. Taylor and E.J. Whitehead: Formal modeling of software architectures at multiple levels of abstraction. Proc. California Software Symp., pp: 28-40, 1996, Los Angeles, CA.

[6] R. J. Allen: A Formal Approach to Software Architecture, PHD Thesis, May 1997, http://www2.cs.cmu.edu/afs/cs/project/able/

[7] D. Bennouar: The Integrated Approach to Software Architecture, IR2008/SAP01, LRDSI Lab, CS Department, The Saad Dahlab University, Algeria, Jan. 2008 (in French),

[8] D. Bennouar, T. Khammaci, and A. Henni: A New Approach To Component's Port Modeling In Software Architecture, ACIT'2007, Lattikia, Syria, Dec; 2007

[9] I. Krechetov, B. Tekinerdogan, , *A.* Garcia, , C. Chavez, , U. Kulesza: Towards an Integrated Aspect-Oriented Modeling Approach for Software Architecture Design. in 8th International Workshop on Aspect-Oriented Modeling, AOSD 2006. 2006. Bonn, Germany

[10] A. Navasa, M. A. Perez, J. Murillo, and J. Hernandez: *Aspect oriented software architecture: a structural perspective*. In Proceedings of the Aspect-Oriented Software Development, 2002, The Netherlands.

[11] J. Aldrich: Open modules : Modular reasoning about advice. In *ECOOP 2005 - Object-Oriented Programming, 19th European Conference,*

Glasgow, UK, July 25-29, 2005,Proceedings, volume 3586, pages 144–168. Springer. 86

[12] R. Norman, F. Marc, S. Scott: JBoss 4.0 - The Official Guide, Sams, April 2005.

[13] D. Suvée, W. Vanderperren, and V. Jonckers: JAsCo: an aspectoriented approach tailored for component based software development. In Proceedings of the 2nd international conference on Aspect-oriented software development, pages 21_29. ACM Press, 2003.

[14] H. Fakih, N. Bouraqadi, and L. Duchien: Aspects and software components: A case study of the fractal component model, International Workshop on Aspect-Oriented Software Development (WAOSD 2004), Beijing, China, September 2004.

[15] N. Pessemier: Unification of aspects and components approaches PhD thesis, université of Science an Technologies at Lille, France (in french), June 2007

[16] R. Miles - AspectJ Cookbook – Real World Aspect Oriented Programming with Java, O'Reilly, 2005

[17] Action semantics for the UML, Final submission. TR, Object Management Group, 2001

[18] D.Bennouar: Aspect Oriented Software Architecture in the IASA Approach, IR2008/SAP02, LRDSI Lab, CS Department, The Saad Dahlab University, Algeria, April 2008.

[19] É. Bruneton, T. Coupaye, M. Leclercq, V. Quéma and J.-B. Stéfani: The fractal component model and its support in java, Software Practice and Experience, special issue on Experiences with Auto-adaptive and Reconfigurable Systems, 2006.

[20] A. Saadi: An action language for the specification and validation of software architecture behavior, Magister Thesis, LRDSI Lab; Computer Science Department, The Saad Dahlab University, Blida, Algeria, June 2008.

[21] N.R. Mehta, N. Medvidovic, S. Phadke: Towards a Taxonomy of Software Connectors, Proceedings of ICSE2000, May 2000.

[22] T. Bures, F. Plasil: Scalable Element-Based Connectors, Proceedings of SERA 2003, SF, USA, June 2003.

[23] J. Aldrich: Using Types to Enforce Architectural Structure, PHD Thesis, Computer Science and Engineering, Washington University, 2003

[24] R.E. Filman and D.P. Friedman, Aspect-Oriented Programming is Quantification and Obliviousness, Workshop on Advanced Separation of Concerns, OOPSLA 2000

[25] John D. Poole: Model-Driven Architecture : Vision, Standards And Emerging Technologies, ECOOP 2001, Workshop on Metamodeling and Adaptive Object Models, April 2001.

[26] D. Bennouar: The design of a complex software system using the IASA software architecture approach, IR2008/SAP03, LRDSI Lab, CS Department, The Saad Dahlab University, Algeria, May 2008.

[27] D. D'Souza and A. Wills: Objects, Components and Frameworks With UML : The Catalysis Approach.. Addison-Wesley, 1999

[28] K. J. Lieberherr: Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns, PWS Publishing Company, International Thomson Publishing, Boston, 1995.