AN ARCHITECTURE-CENTRIC APPROACH FOR MANAGING THE EVOLUTION OF EAI SERVICES-ORIENTED ARCHITECTURE

Frédéric POURRAZ, Hervé VERJUS
LISTIC, University of Savoie, France
frederic.pourraz@univ-savoie.fr, herve.verjus@univ-savoie.fr

Flavio OQUENDO
VALORIA, University of Bretagne-Sud, France
flavio.oquendo@univ-ubs.fr

Keywords: EAI architecture, evolution, ADL, SOA, web services, architecture-centric approach.

Abstract: The development of large software applications (like EAI solution) is oriented toward the interoperation of existing software components (like COTS and legacy systems). COTS-based systems are built in ad-hoc manner and it is not possible to reason on them no more it is possible to demonstrate if such systems satisfy important properties like Quality Of Service and Quality Attributes. On the other hand, software architecture domain aims at providing formal languages for the description of software systems allowing checking properties (formal analysis) and to reason about software architecture models. The paper proposes an approach that consists in formalizing, deploying and evolving EAI architectures. For that purpose, the ArchWare environment and engineering languages (especially the ArchWare formal ADL, based on the π-calculus) and accompanied tools are used.

1 INTRODUCTION

Information systems are now based on aggregation of existing components that have to cooperate in a precise manner in order to satisfy user needs and software functionalities.
A new technology has emerged consisting in assembling widely distributed services for building a services-based application, by the way of web standards such as SOAP, XML, WSDL, etc. A set of interacting (Web) services is known as a Services-Oriented Architecture (SOA). One of the main features of SOA is that services involved are autonomous (we will discuss such features latter in this paper) are widely distributed across the Internet.
Information systems are more and more complex, need more and more functionality provided by several software applications that already exist (COTS or legacy systems). Reusing and assembling existing components (COTS or/and legacy systems) are questions that cope with some difficulties that are not covered by classical component-based programming solutions like EJB, COM+, CCM, etc. As these are specifications for components development, they do not address the case of COTS-based systems, where source code is not available or/and has been previously developed with other specifications and programming languages. The EAI (Enterprise Application Integration) domain provides integration models and techniques for assembling heterogeneous software applications in a pragmatic way. EAI emerging solutions encompass (1) a distributed architecture using web services and (2) a description of the web services centric architecture, expressed using a web services orchestration/choreography language (i.e. XLANG, WSFL, BPEL4WS, etc.). Information systems based on such technology integrate heterogeneous software components, COTS, using a process-based integration approach, where the process description has to insure the execution correctness of the system. Such information systems, building from COTS, will be called COTS-based systems in the following.

In such context, an issue is still open: the adequation between the information system provided (i.e. its composition) and the functionalities it would able to provide (i.e. to the end user) and the orchestration of such functionalities according to business processes. Because EAI solutions fail in insuring that the information systems provided succeeds in end-user needs satisfaction, we need new approaches.
This paper presents our work in formally describe an EAI solution building from COTS (or legacy systems). The approach used is based on an architecture-centric development process where the system description is the heart of the process. Using such approach, the (abstract) description can be checked, refined in order to obtain more concrete enactable descriptions. The paper will also show how architectural evolution is supported. We assume in this paper that COTS are able to interoperate using web services.
The paper will first present (section 2) the business case in a European project context. In section 3, we will introduce the formal approach we follow for defining, enacting and evolving EAI architectures. The, section 4 will present the formalization of such architectures, especially from the evolution point of view. Deployment and enactment of the generated services oriented architecture will be discussed in section 5, while section 6 will conclude.

2 BUSINESS CASE: EAI ARCHITECTURE FOR AGILE ENTERPRISES

2.1 Business case scenario

Our business case relates to a company that manufactures a specific product (an axis – a mechanical piece). Moreover, this company can subcontract a part of its manufacturing to a subcontractor in order to carry out a specific task.

As you know, enterprises (especially Small and Medium Enterprises) have to be very adaptive in order to satisfy market changes, business changes, customer's needs, etc. Agility stands for an enterprise being able to quickly change, adapt their processes according to market changes, without process, end-user and customer service interruption. In such way, the EAI architecture has be able to change dynamically taking account new requirements and new business contracts.

Agility interests are manifold and covers some topics among them (Bland dit Jolicoeur et al., 2002):

- Increase of compliance with ISO 9001 V 2000 within and between SMEs due to ability to identify business processes;
- Competitiveness increases due to ability to control and adapt business processes;
- Productivity increases due to ability to synchronize business processes (e.g. in 'just-on-time delivery' in the automotive industry) and, as a consequence, increase quality of relationship between SMEs;
- Increase of faith in information system results due to ability to proof business processes properties;
- Investment costs reduction due to ability to integrate existing applications (legacy software or specific applications).

We adopt an illustrating evolution scenario in EAI context. In the scenario, several companies COTS (like ERP, production follow-up software and SPC) are involved thanks to a choreographer, which will orchestrate them according to the wished business process (see figure 1 (a)):

- An ERP (called Infodev) will first send a manufacturing order to the choreographer, which will deploy the different operations in production.
- The operations list will be send to a production follow-up software (called Alpha3i) and machines to controlled to a SPC software (called Arve).
- The scrap number per operation will be returned to the choreographer, which will next pass this number to the ERP.
- In parallel, a quality control will be performed by the SPC software. We can underline that this operation don't send any information to the choreographer, it's a "vertical" application.

Following a planning step on the Contractor level, an overload is detected; this overload is transmitted to the choreographer which starts a process of invitation to tender with various subcontractors. In this case, the previous scenario needs to evolve by adding new business process functionalities in the choreographer (see figure 1 (b)):

- Within this evolution, the choreographer will receive an overload (higher than 50 pieces in our case) manufacturing order from the ERP (named Infodev).

- Then the choreographer will execute in parallel the same scenario as previously described for an order of 50 pieces and a subcontracting process, which implies a new functionality (a negotiation service) to the architecture. This new service consists in finding the best supplier in order to manufacture the production overload.

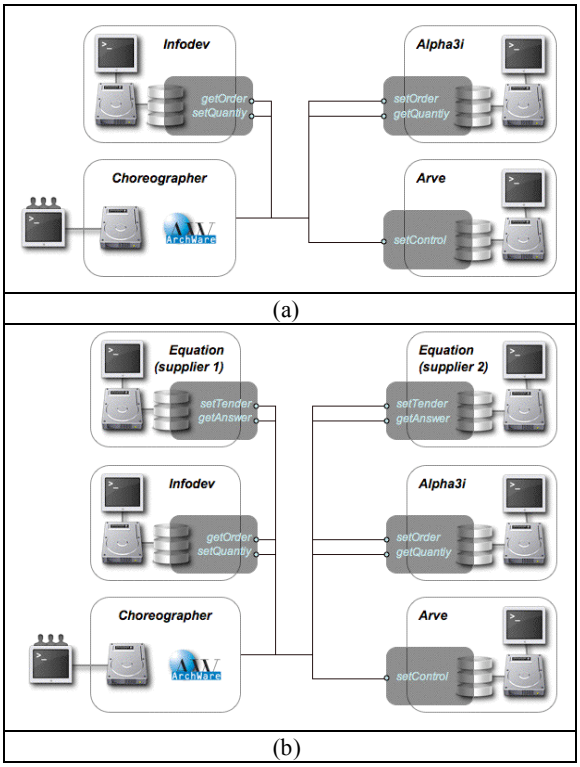

Figure 1: EAI COTS architectures.

2.2 EAI engineering issues

To meet such requirements according to the evolution scenario, best-suited engineering approaches have to be chosen.
In the case of complex COTS-based systems (like enterprise information systems – in our case EAI) usual and classical engineering approaches fail:

- SMEs need systems that are adapted to their requirements: the design (including properties) of such systems is a crucial step but systems designs/models have to be validated before implemented (Blanc dit Jolicoeur et al., 2002).
- COTS are specific software components with which components classical integration patterns or idioms are not relevant (Estublier et al., 2001) and (Cimpan et al. 2003): COTS have to be characterized as well as their integration.

COTS-based system models (when existing) cannot be checked nor validated. That is, one cannot reason on models nor analysis can be made on such models. This lack of formalization has following consequences:

- Design (of complex systems) expertise cannot be caught nor maintained;
- There is a gap and discrepancies between the design and the execution. It is impossible to guarantee that the execution will be conformant to the design;
- The COTS-based systems evolution (substitution, deletion, addition of COTS, changing system behaviour, process, etc.) is not well supported nor it can be validated;
- Crucial properties (safety, completeness, consistency, etc.) of the systems are not taken into account.

As EAI solutions are located in a distributed context (distributed enterprises, networked enterprises, …), the EAI architects are enthusiastic by using web services as technology for supporting COTS interoperability. SOAs have to deal with many of the issues encountered in more classical COTS-based systems. Web services will be employed as COTS facets (also called wrappers) that will be orchestrated in order to satisfy EAI goals (the production a mechanical piece: an axis in our case). Web services are accompanied by standards that support part of the interoperability (i.e. SOAP protocol, WSDL, etc.).

We propose to follow an architecture-centric approach taking into account evolution and proposing some solutions that might meet the needs issued by the previously identified limitations.

3 A SOFTWARE ARCHITECTURE-CENTRIC APPROACH FOR FORMALIZING EAI SOA

3.1 Architecture-centric approach

The architecture of software intensive system (such as an EAI architecture) defines the elements that compose the system, and how they interact. The software architecture definition can be made informally, or by using a dedicated language. Different abstraction levels are considered for describing the software architecture (Allen and

Garlan, 1997). The use of formal architecture refinement guarantees the preservation of properties specified at abstract levels all the way towards architecture implementation.
The architecture centric development process (see figure 2) aims at providing means for defining software intensive systems at a very abstract level. Such descriptions can be then validated in order to check systems properties and are refined in a more concrete description (that allows to deploy the system in a concrete environment).

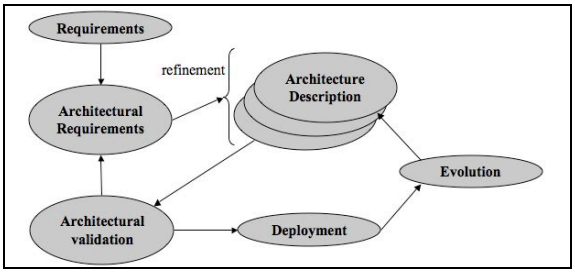

Figure 2 - Architecture centric development process

The architecture-centric development process we propose (see figure 2) is quite different to the classical software development process (i.e. waterfall, spiral, etc.): if the system behaviour does not fit the requirements, the architecture description can be modified without restarting entirely the development process. Representations (architectural descriptions) are also checked at every stage of the process before generating the code.
The work on architecture centric approaches for software development has been very fruitful during the past years, leading, among other results, to the proposition of a variety of Architecture Description Languages (ADLs), usually accompanied by analysis tools. The enthusiasm around the development of formal languages for architecture description comes from the fact that such formalisms are suitable for automated handling. These languages are used to formalize the architecture description as well as its refinement. The benefits of using such an approach are manifold. They rank from the increment of architecture comprehension among the persons involved in a project (due to the use of an unambiguous language), to the reuse at the design phase (design elements are reused) and to the property description and analysis (properties of the future system can be specified and the architecture analyzed for validation purpose).
Different ADLs have been proposed (Medvidovic and Taylor, 2000). According to our requirements we need a formal ADL that will support: the description of the architectural structure and behaviour, the properties expression and dynamic evolution. We also need a way to express processes. We will introduce the ArchWare approach that combines interesting formal features in a unified ADL.

3.2. The ArchWare environment

The main objective of the ArchWare project (European IST-5 project, number 32360) was to provide the necessary elements to the engineering of evolvable software systems. In order to achieve this goal, the ArchWare project developed an integrated set of languages and architecture centred tools while being based on a persisting execution framework (Morrison et al., 2004).
The ArchWare project provides some engineering technologies, among them:

- Innovating languages centred architecture (architecture description language and analysis language),
- Refinement models,
- Customizable software environments and tools dedicated to the engineering of evolutionary software systems.

ArchWare aimed at building a customizable environment of engineering software, which can be used to create software architecture centred environments (Oquendo et al., 2004). This project considered that a customizable architecture centred environment is structured in two distinct layers, namely a runtime framework and a set of architecture centred tools.
The ArchWare runtime framework (Morrison et al., 2004) includes an execution engine of architectures based on evolutionary processes of development, a refinement process of architecture description and mechanisms supporting the interoperability of the environment tools and components (that can be COTS). Details of the ArchWare environment can be found in (ArchWare consortium, 2001).
The ArchWare architecture centred tools provides supports for:

- The definition of the architecture,
- Validation of such architectures (using analysis tools and software graphical animation tool),
- The checking of the functional and extra functional properties of architectures,
- The refinement of architecture descriptions from an abstract level to a concrete level,
- The code generation of the systems in various programming languages (using explicit rules).

3.3. Architecture evolution support

One of the ArchWare environment key features is the evolution support ability (Cimpan and Verjus, 2005). On one hand, ArchWare ADL is the language allowing to describe evolvable architectures (i.e. architectures that can dynamically evolve); on the other hand, the ArchWare environment contains an ADL virtual machine (Morrison et al., 2004) that supports dynamic evolution (the architecture description code can be modified while being interpreted). Then, an architecture description can be dynamically changed and the runtime architecture change accordingly (we will present an evolution scenario latter in this paper). When an architecture evolves dynamically, one may check the new architecture against properties or not (it is up to the architect).

According to our needs, the ArchWare ADL (Oquendo et al., 2002) is the only one formal ADL (section 2) that :

- Allows the architecture structural modelling as well as the behavioural description (as an extension of π-calculus (Milner, 1999));
- Supports properties/constraints definition;
- Supports dynamic evolution of the architecture.

We will also propose a way to enact services oriented architecture as the concrete EAI architecture, following the refinement approach.

4 EAI ARCHITECTURE EVOLUTION DESCRIPTION

We decided to describe EAI architectures using ArchWare ADL. According to the architecture-centric process we adopt (see figure 2), the formal descriptions can then be refined in order to obtain a concrete representation (figure 1). At the early stage of the development process (figure 2), abstract architectures have to be expressed according to domain specific characteristics and ilities.

The EAI architecture description contains the code which represents the first stage of the architecture (figure 3), with the code allowing the architecture to request an evolution in order to behave as the architecture shown in figure 4 The evolution has also to be expressed at the architectural level (the evolution is described using the ArchWare ADL code – it is part of the EAI architectural description) in order to deal with the evolution that occurs at the enterprise level.

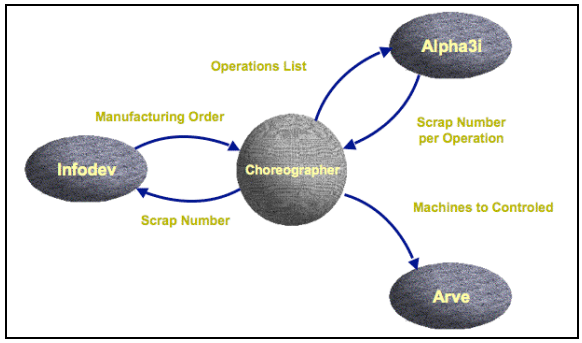

Figure 3: Business case EAI architecture

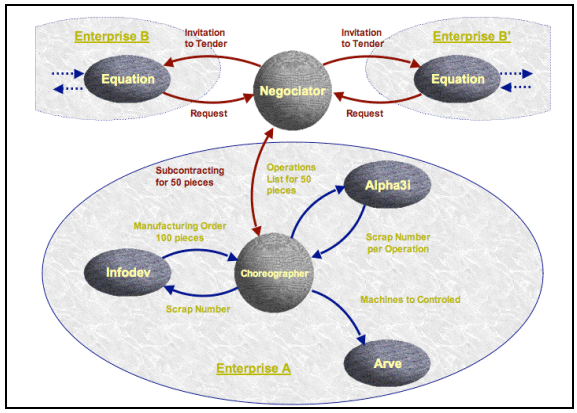

Figure 4: EAI architecture after evolution

Main pieces of EAI architecture description code (in ArchWare ADL) including evolution expression are presented hereafter. The code contains both the classical architectural description in terms of elements (often called processes or components in ADL) interacting each other, some properties according to EAI domain and business process code. It is innovative as the ArchWare ADL language allows to formalize several facets of an EAI architecture (topology, properties, business process and evolution that can focus on part or all of the architectural artefacts).

The ERP COTS (Infodev) can be defined in ArchWare-ADL as follow:

```
value erp is abstraction(); {
  value getOrder is free connection(String,
  String, Integer, String);
  via getOrder send "order-1", "axe", 100,
  "JUN 17 2005";
  value setQuantity is free connection
  (String, String, String, Integer);
  via setQuantity receive store:String,
  code:String, article:String,
  quantity:Integer;
  done }
```

According to ArchWare ADL concepts (Oquendo et al., 2002), each COTS is described as an abstraction which declares some connections on where parameters can be sent and/or received.

The ERP COTS has got two sequential actions. First, it sends the description of the new order via the "getOrder" connection. Secondly, it will receive the production report via the "setQuantity" connection.

According to the ArchWare ADL syntax, the keyword "done" stands for the terminate action (*Tau* in π-calculus).

As well as the ERP COTS, both production report and SPC COTS are designed by using abstractions. As previously described, they define some connections allowing them to send and to receive parameters.

The definition of the choreographer is also based on an abstraction description but contains more complicated actions. After receiving an order via the "getOrder" connection, the choreographer composes two processes in parallel:

1. The first one is the internal manufacturing process. It first calculates the right quantity to product (100 or the initial quantity if it is less than 100). After that, it sends the order to product and to control (via setOrder and setControl), receives the production report and transmits it to the ERP (via getQuantity and setQuantity);
2. The second one requests an evolution in case of overload detection. It first tests the quantity and if an overload is found, it will send a request to a particular ArchWare tool: Hypercode-Editor (Morrison et al., 2004) (via hypercode_request). The end-user architect is asked to provide a new abstraction that corresponds (its behaviour) to the subcontracting process (see section 5). This latter is received (remember that one of the powerful features of the π-calculus is that processes can be exchanged between other processes) and instantiated by the choreographer (via hypercode_reply).

```
value choreographer is abstraction(); {
  value getOrder is free connection(String,
  String, Integer, String);
  via getOrder receive code:String,
  article:String, quantity:Integer,
  date:String;

  compose {
    behaviour {
      if(quantity > 50) then {
        value newQuantity is 50; }
      else {
        value newQuantity is quantity; }
      value setOrder is free connection
      (String, Integer, String, String);
      value setControl is free
      connection(String);
      via setOrder send code, quantity,
      article, date;
      via setControl send code;
      value getQuantity is free
      connection(String, String, Integer);
      via getQuantity receive code:String,
      article:String, quantity:Integer;
      value setQuantity is free connection
      (String, String, String, Integer);
      via setQuantity send "stock", code,
      article, quantity; }
    and behaviour {
      if(quantity > 50) then {
        value hypercode_request is free
        connection();
        via hypercode_request send;
        value hypercode_reply is free
        connection(abstraction(String,
        Integer));
        via hypercode_reply receive
        subcontracting_process:
        abstraction(String, Integer);
        subcontracting_process(article,
        quantity-50); } } } }
```

Finally, the following EAI abstraction that corresponds to the bootstrap of the EAI architecture must be defined. This abstraction instantiates, in a single process, the COTS abstractions previously defined and unifies all of the connections.

```
value eai is abstraction(); {
  compose {
    choreograher() and
    erp() and
    production() and
    spc()
    where {
      choreograher::getOrder
      unifies erp::getOrder
      and ... } } }
```

Once defined, the EAI architectural description can be analyzed (see figure 2). Then, the architecture is deployed as a services oriented architecture and can be interpreted by the ArchWare runtime environment. One can note that we are now able to easily adapt such architecture for other EAI solutions (by modifying ArchWare ADL code).

The next section will show the refinement consisting in generating web services WSDL code from a ArchWare ADL specification,

5 DEPLOYING, ENACTING AND EVOLVING EAI ARCHITECTURE AS A SOA

5.1 Architecture deployment and execution

At the final stage of the refinement (the concrete architecture), we obtain a Services Oriented Architecture where web services are used as COTS facets. The web services allow EAI components (COTS and legacy system) to interoperate (figure 3). In such concrete context, all well-known languages (WSFL, XLANG, BPEL4WS, etc.) and web-based technologies (WSDL, SOAP, etc.) may be candidates for supporting the deployment and the execution of our systems using web services.

In our particular case, the business process is part of the entire architecture and expressed using ArchWare ADL (the choreographer is an abstraction in term of ArchWare ADL and its behaviour is the business process).
As we introduced previously, the ArchWare runtime environment enacts the entire architecture (including business processes). External components (COTS) will interoperate with the ArchWare environment through web services while the choreographer will be part of the ArchWare environment (enacted by the ArchWare virtual machine).
Due to these considerations, from the architectural abstract description we generate the COTS web services concrete description (the WSDL code is basically obtained from our abstract description of services the COTS provide - their APIs and only if the web services do not exist). Generic refinement rules presented hereafter support the transformation from ArchWare ADL to WSDL. All other elements of the abstract architecture are enacted by the virtual machine.

First rule

When a new connection is declared followed by a send action on it, WSDL representation can be made by adding news <wsdl:message> tags, one for the request and one for the response. In this case, the request corresponding tag is empty whereas the response corresponding tag defines a parameter which can be a simple type or can point towards a new type definition (<wsdl:types> tag) where a complex type is design (<complexType> tag).

```
value getOrder is free connection(String,
String, Integer, String);
via getOrder send …
```

```
<wsdl:types>
  <schema>
    <complexType name="Order">
      <sequence>
        <element name="param1" type="soapenc:string"/>
        <element name="param2" type="soapenc:string"/>
        <element name="param3" type="soapenc:integer"/>
        <element name="param4" type="soapenc:string"/>
      </sequence>
    </complexType>
  </schema>
</wsdl:types>
<wsdl:message name="getOrderRequest"/>
<wsdl:message name="getOrderResponse">
  <wsdl:part name="param" type="tns:Order"/>
</wsdl:message>
```

Second rule

When a new connection is declared followed by a receive action on it, WSDL representation can be made by adding news <wsdl:message> tags, one for the request and one for the response.
In this case, the response corresponding tag is empty whereas the request corresponding tag defines simple type parameters.

```
value setQuantity is free connection
(String, String, String, Integer);
via setQuantity receive …
```

```
<wsdl:message name="setQuantityRequest">
  <wsdl:part name="param1" type="soapenc:string"/>
  <wsdl:part name="param2" type="soapenc:string"/>
  <wsdl:part name="param3" type="soapenc:string"/>
  <wsdl:part name="param4" type="soapenc:integer"/>
</wsdl:message>
<wsdl:message name="setQuantityResponse"/>
```

Other rules

On the preceding model, ArchWare-ADL description allows to define the global <wsdl:definitions> tag as well as <wsdl:portType>, <wsdl:operation>, <wsdl:binding> and <wsdl:service> tags. All these rules won't be define in this article.

5.2. Architectural dynamic evolution

The concrete architecture can now be interpreted. According to the business scenario we presented in section 2, the EAI architecture is behaving as the one shown in figure 1(a). Then, when a production capability threshold is reached, suppliers have to be added in order to satisfy the new production demand. Then, the EAI architecture is now the one presented in figure 1(b) and is behaving as the latter. The evolution concerns:

- The architecture topology (by addition of suppliers – several abstractions in ArchWare ADL concepts);
- The communication between architectural elements (i.e. connections between abstractions);
- A new business process with several enterprises and more COTS involved.

Note that the concrete architectures (SOAs) presented in figure 1 are not symmetric (in term of number of architectural elements) to the abstract architecture detailed in section 4 (figures 3 and 4). This is due to that more architectural elements (abstractions) are necessary to be defined in order to provide functional and non-functional architectural aspects required by EAI architectures. The concrete architectures are only composed by all of the required web services (one per COTS) plus the ArchWare environment that enact the architecture (both the enacted architecture and the ArchWare architecture centred tools).

6 CONCLUSION AND ONGOING WORK

Building COTS-based system generally fails due to non-formal approaches (often ad-hoc solutions like in classical EAI engineering approaches) used. In (Estublier et al., 2001) and (Verjus et al., 2002) we claimed that designing and building COTS-based systems addresses lots of issues: the gap between the design level and the implementation one and the evolution support are two of them. Because COTS

(as well as legacy systems) already exist, we "only" have to deal with the "glue" between such software components (COTS, etc.). The approach presented in this paper innovates by providing a formal approach for the development, deployment and enactment of an EAI architecture as well as its dynamic evolution (Cimpan and Verjus, 2005). This approach combines an unified approach consisting in refinement steps from specification to implementation code generation, and a more pragmatic approach for which we only focus on the "glue" that have to guarantee the properties of the COTS-based system the architect is interested. Our approach is divided in two parts: (1) the definition of an architecture that is convenient for the design of COTS-based systems as well as it is also closed to a concrete architecture (in our case, a SOA) and (2) an architecture-centric development process using a formal ADL as a specification language (that deals with structural aspects and behavioural aspects - including business processes).

This approach has been validated in the European ArchWare project and in an R&D project with SMEs and manufacturing companies. This work is now continued in order to provide a formal Domain Specific Language for describing generic SOA in terms of formal architectural constructs.

Some works focus on business process description (van der Aalst et al., 2003) in SOA, mostly using XML-based languages (such as BPEL4WS (Curbera et al., 2003), WSFL (Leymann, 2001), etc.); some other focus on services semantic (McIlraith et al., 2001) description (i.e. OWL-S (OWL, 2003), WSMO (Priest and Roman, 2004), etc.) for services discovery, selection and composition. Some consortiums, projects (i.e. SWSI, Knowledge Web), aim at addressing all of the SOA facets. Interesting results are expected. As far from now, such works do not address formal description (van der Aalst et al., 2003) from an architectural point of view (where architectural ilities and constraints checking and validation are supported), nor they cover evolution.

REFERENCES

Allen, R. and Garlan, G., 1997. A formal basis for architectural connection. *ACM Transaction on Software Engineering and Methodology*.

Blanc dit Jolicoeur, L., Braesch, C., Dindeleux, R., Gaspard, S., Le Berre, D., Leymonerie, F., Montaud, A., Chaudet, C., Haurat, A., Théroude, F., 2002. Final Specification of Business Case 1, Scenario and Initial Requirements. *Deliverable D7.1b, ArchWare project.*

Estublier, J., Verjus, H., Cunin P.-Y., 2001. Building Software Federation. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Las-Vegas, USA.

Verjus, H., Cîmpan, S., Telisson, D., 2002. Formalising COTS-based federations using software architectural styles. *In Proceedings of the 15th International Conference Software & Systems Engineering and their Applications*, December 2-5, Paris, France.

Oquendo F., Alloui I., Cîmpan S., Verjus H., 2002. The ArchWare ADL: Definition of the Abstract Syntax and Formal Semantics. *ArchWare European RTD Project IST-2001-32360*, Deliverable D1.1b, December.

Cimpan S., Verjus H., Oquendo F., 2003. COTS-based System Design using Software Architectural Styles, *Integrated Design and Process Technology (IDPT'03),* Austin, USA, December, pp. 127-134.

Cimpan S., Verjus H., 2005. Challenges in Architecture Centred Software Evolution. *CHASE Workshop: Challenges in Software Evolution*, Bern, Switzerland.

Oquendo F., Warboys B., Morrison R., Dindeleux R., Gallo F., Garavel H., Occhipinti C., 2004. ArchWare: Architecting Evolvable Software. *In proceedings of the first European Workshop on Software Architecture*, pages 257-271, St Andrews, UK, May.

ArchWare Consortium, 2001. The EU funded ArchWare – Architecting Evolvable Software - project : http://www.arch-ware.org

Robin Milner, 1999. Communicating and Mobile Systems: the pi-calculus. *Cambridge University Press*.

Morrison R., Balasubramaniam D., Kirby N.C., Mickan K., Oquendo F., Cimpan S., Warboys B., Snowdon R., Greenwood M., 2004. Support for Evolving Software Architectures in the ArchWare ADL. *4th Working IEEE/IFIP Int. Conf. on Software Architecture,* Oslo, Norway, June, pp. 69-78

Medvidovic, N. and Taylor, R.N., 2000. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, January.

McIlraith, S., Son, T., and Zeng, H., 2001. Semantic web services. *IEEE Intelligent Systems,* 16(2).

van der Aalst, M.P., Dumas, M., ter Hofstede, A.H.M., 2003. Web Service Composition Languages: Old Wine in New Bottles? *In Proceedings of 29th IEEE Euromicro Conference,* p. 298.

Leymann, F., 2001. Web Services Flow Language (WSFL 1.0).

Curbera, F., Goland, Y., Klein, J., Leyman, F., Roller, D., Thatte, S., Weerawarana, S., 2002. Business Process Execution Language for Web Services (BPEL4WS) 1.0. *http//www.ibm.com/developerworks/library/ws-bpel.*

OWL Services Coalition, 2003. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.0/owl-s.pdf

Priest, C. and Roman, D., 2004. Web Service Modeling Ontology - Full (WSMO - Full). http://www.wsmo.org/2004/d12/.