FOCAS: An Enginering Environment for Service-Based Applications

Gabriel Pedraza, Idrissa A. Dieng and Jacky Estublier

Grenoble University - LIG, 220 rue de la Chimie, 38041 Grenoble BP53 Cedex 9 France

Abstract. Service composition is an important topic, but so far addressed from a technical and low level perspective. The issue is not (too much) the orchestration formalism, but rather the engineering issues related to the many concerns that must be combined, to the technical complexity, to the heterogeneity and incompatibilities between available services, and the low level formalism and tools. The paper presents FOCAS, a full fledge environment that targets the different categories of stakeholders involved in the design, development, and maintenance of a service-based application. FOCAS first separates the different concerns, both functional and non functional, that made up a service-based application; second, it separates different levels of abstraction, and third it establishes links and mediations between these concerns and levels of abstraction. The paper presents FOCAS its principles, implementation and its evaluation.

1 Introduction

To face size and increase of complexity, and to meet demands for cost and time to market reduction, Software Engineering, since the 70s, relies on very few principles: information hiding and reuse. Information hiding states that "clients" of a part should ignore most of what is inside the part; only the very minimal amount of information should be made visible and accessible to the client, most often called its interface. The available technologies are different ways to follow these principles and parts have been successively called module, class, component and service. Service is the latest evolution in the direction of making independent parts and to improve composition flexibility. For that reason, Service Oriented Architecture (SOA) is the de facto standard for dynamic applications.

Recently, the separation of concerns paradigm was introduced as a special way for defining parts with very little dependencies. Unfortunately, there is no general approach that helps the designer to find out how to decompose an application along different concerns, nor how to combine them in order to ensure the correct execution of the resulting application. The experience even shows that finding the "right" concerns, and defining how to combine them in order to build advanced applications is a major difficulty, hampering in practice the deployment of the separation of concerns paradigm in the software industry.

Pedraza G., Dieng I. and Estublier J. (2009).
FOCAS: An Enginering Environment for Service-Based Applications.
In *Proceedings of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering - Evaluation of Novel Approaches to Software Engineering*, pages 21-32
DOI: 10.5220/0001952700210032

Both SOA and separation of concerns are extremely flexible and rather new, therefore the design and development of industrial SOA applications using different concerns face many Software Engineering challenges as: how to structure the application in parts (services), what are the relevant concerns, and how to compose the parts and the concerns.

In this paper, we present FOCAS a framework that borrows ideas from the SOA and separation of concerns paradigm in order to support designers and developers that want to build service-based applications.

The paper is organized as follows. Section 2 briefly introduces the service oriented architecture and the orchestration approach. Section 3 outlines our approach, describes its principal components, and its extension capabilities. Section 4 presents the FOCAS environment. Section 5 describes the validation and implementation of our approach. Section 6 relates our work to the existing literature. Finally, section 7 concludes the paper.

2 Service Oriented Architecture

Service Oriented Architecture (SOA) improves significantly the decoupling between parts (services), but faces composition issues: what is the decomposition and composition process, how to handle the different dynamic behaviour (service life-cycle, data and interface (un)availability and (in)compatibility). In SOA, not only composition is dynamic, but some services platforms like OSGi [1] allow dynamic addition and removal of service instances at runtime.

Web services technology is one of the most successful SOA technologies. Web Services are software systems which support machine-to-machine interactions over a network based on a set of standards [2]: WSDL for service description, and XML and SOAP for representation and communication. Web services extensions include semantic description (in WSML and OWL [3] languages), semantic matching of services [4] or invocation protocol specification in WSCI.

Services Orchestration. Service orchestration is an approach to build (service-based) applications by defining services execution order (control flow) and their data exchange (data flow). In orchestration, the control is explicit, centralized and external to services.

Several orchestration languages have been designed, among them XLANG [5], WSFL [6], BPML and Business Process Execution Language for Web Services as short WS-BPEL [7] [8]. WS-BPEL, created by Microsoft, IBM, SAP and others, is the de facto standard in web services orchestration.

In WS-BPEL, parts are called activities that can be basic or structured. Basic activities are used for service interaction, structured activities are control operators close to classic programming languages; therefore, WS-BPEL orchestrations are modeling a programmatic solution to a problem, not the problem itself. WS-BPEL is designed only for Web Services orchestration and it cannot orchestrate other kinds of services (like OSGi). In addition, WS-BPEL does not have human interaction capabilities and cannot easily perform computation. Finally, WS-BPEL process

models are monolithic which makes difficult to achieve their evolution and reuse capabilities.

These problems have been acknowledged, and many extensions have been proposed, such as BPEL4J [9] which combines Java code with WS-BPEL orchestrations or BPEL4People [10] which adds process interaction with human beings; but these "solutions" are partial and difficult to implement. Finally, WS-BPEL orchestration models are strongly coupled with the WSDL definition of the used web services. To use another service providing the same functionality, described in another WSDL interface, one must change the process definition.

3 Our Approach

Our work reconciles, in a single framework the separation of concerns paradigm, SOA technologies and the classic Software Engineering approaches, in a software engineering environment called FOCAS (Framework for Orchestration, Composition and Aggregation of Services).

FOCAS defines four layers: Business, Abstract Orchestration, Mediation, and Concrete Services. The business layer defines the problem to solve as a process controlling evolution of the business concepts. The abstract orchestration layer defines orchestration as a possible implementation of the business process in term of abstract services, additional behavior, human interaction or other software components. The actual web services and their data are provided in the concrete services layer. If the concrete and abstract services have been designed independently, it is likely that they do not match, in that case the mediation layer is used to solve (some) data or service incompatibilities.

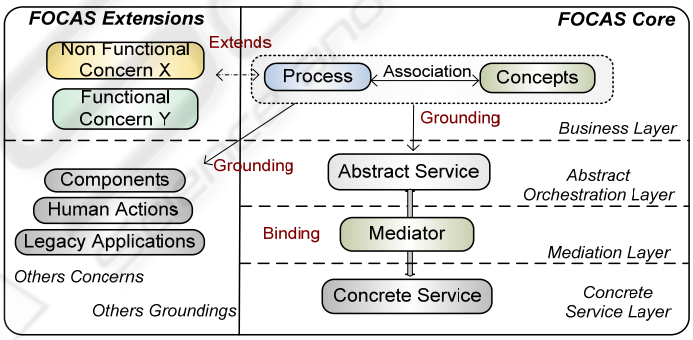

Fig. 1. FOCAS General Architecture.

FOCAS proposes a comprehensive environment in which these layers and concerns are managed, automating concerns "weaving", making explicit and supported service composition and mediation, which altogether turns service composition easy and accessible to non experts. FOCAS Core, topic of this paper, emphasizes Web Services, but its extensions, used in various industrial applications, applies the same approach to humans and other kind of executable components. 0

presents the FOCAS general architecture, based in the layers commented above, and in its extension capabilities.

In this section the traditional simplified flights reservation example will be used to illustrate our approach.

3.1 The Business Layer

This layer is intended to define, through a process model, a given business activity (e.g. selling a flight ticket). It expresses the activities acting upon entities pertaining to the business domain (customers, flights and so on), to perform a business action.

This layer separate two concerns:

- The data, entities associated with the business concepts. Data concern expresses the structure of these entities.
- The overall logic of the business actions expressed as a process made of activities that create and changes the business data.

FOCAS follows the “external control” paradigm which emphasizes the process logic (the orchestration approach) against the entity behaviour (the O.O. approach). The global control logic is defined in the process, and no or limited semantics is expressed in the business (data) objects. This separation of concerns allows the same business concepts to be managed by different process, and the same process to be performed on different data as long their intention is the same (e.g. *Customer* v.s. *Client* v.s. *Passenger Types*), irrespective of a given execution technique.

FOCAS clearly separates a business layer, in which is defined the business process, from an orchestration layer, in which is defined how the business process is implemented in term of services. The separation is a major change with respect to usual orchestration systems which define the execution ordering of existing services, not the application business logic.

The Abstract Process Engine Language APEL is used to express the process model in our approach [11]. APEL is a high level process definition language containing a minimal set of concepts that are sufficient to understand the purpose of a process model.

The main concept is the activity which is a step in the process, and it results in an action being performed. The actual action to be performed is not defined into the process model, and it can be a service invocation (a Web Service, DPWS service, an OSGi service), any kind of program execution (legacy, COTS) or even a human action. Activities in APEL can be composed of sub-activities, this property permits to deal with different abstraction levels in a model. Ports are the activity communication interface; each port specifies a list of expected products.

A Product is an abstract object (records, data, files, documents, etc) that flows between activities. Products are represented by variables having a name and a type which are only symbolic names (e.g. “client” is a product of type “Customer”). This property does not preclude of the actual nature, structure and content of the real data that will circulate in process. Dataflows connect output ports to input ports, specifying which product variables are being transferred between activities.

APEL has a graphical concrete syntax, an activity (from outside) is represented as a rectangle with tiny squares on sides which denotes its ports. Internally an activity is

represented as a rectangle containing sub-activities, and its ports are represented as triangles. This dual representation is used to navigate across a complex model composed of several levels of embedded activities. Finally a dataflow is represented as a line that connects ports, and products are labels on dataflows.

The business process model of the flight reservation example is created with our graphical APEL editor as illustrated in 0: Initially, the customer carries out a flight search using the *FindFlight* activity. Then, the customer does his/her reservation with the *DoReservation* activity. The customer must confirm the reservation before a given date, otherwise the process sends an electronic mail using the *SendNotification* activity and the process finishes. If confirmed, the customer pays the ticket using the *Payment* activity and the process finishes.

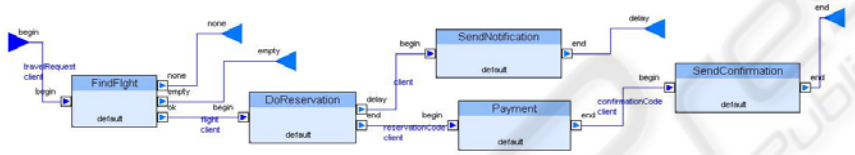

Fig. 2. Business Process Model Example.

In the other hand, the business entities are a structural data representation of the concepts relevant in the domain. In our example, they are *Customer*, *Flight*, *CreditCard*, etc. These entities structure can be also defined in a specialized editor.

3.2 Abstract Orchestration Layer

The abstract orchestration layer is a “grounding” (i.e. the action to be performed) of the business process in term of human actions and/or executable elements [12]. These executable elements can be of different natures: legacy or commercial components, specific executable code or services (Web services, OSGi services, DPWS, etc). FOCAS allows all these possibilities but FOCAS core emphasizes and explicitly supports the last two: specific executable code, called here behaviour, and abstract services definition in order to hide heterogeneity of different implementation technologies.

The behaviour model is a Java snippet code associated with activities which gives very much flexibility to the system. It allows defining directly computation performed by an activity or performing some data transformation before calling services or external programs, whether legacy or commercial, local or remote. It also allows adapting a business process to a given context.

Services provide functionalities that the application will require at execution. An abstract service is a service description, represented by a Java interface in which the methods represent services operations, the parameters being instances of the data entities classes. In our example, two abstract services are defined: *AirlineService* and *BankService*.

An abstract service is only a service description, therefore for an abstract orchestration to become executable, each abstract service must be associated with concrete service(s). “Binding” is the association between an abstract service and a real

service through the mediation layer. Binding in our tool is possible at design time, deployment time or even at runtime (if not mediation is necessary).

3.3 Mediation Layer

The semantic distance between abstract and real services and the fact that they may have been defined independently makes it is likely that there is no direct match between abstract and real services.

If there exists semantic gaps between abstract and concrete services, components charge to bring out this distance can be used in our architecture. These components can even be implemented as small lower level orchestration using our technology, but in this case without using abstract descriptions, instead using directly service implementations.

Even if a real service provides the required functionality, there is still a risk of discrepancies, since abstract and real services can use different names (operation and data types) and different data formats to represent the same computation on the "same" entities. For example, the *Customer* concept in the abstract orchestration may be mapped to the *Passenger* data type in the *Air France* reservation Web service. If the formats are not identical, it will be necessary to solve operation and data interoperability problems.

We have introduced semantic mediators, which can be low-level orchestrations to adapt an abstract service to a set of real services, and data mediators to adapt the business concepts to the concrete services data. With this layer, business models are independent from the actual services and data definition, improving their reusability and evolution capabilities.

3.4 Concerns Extension

The FOCAS platform "natively" supports the business process, business entities (process data), behaviour, mediators, and abstract and concrete services concerns. However, many other concerns are relevant in a real service-based application. FOCAS provides mechanisms to extend the platform with other concerns, either functional or not.

Additional functional concerns are integrated in FOCAS through meta-links between the process concepts and the new functional area concepts, in exactly the same way as we have defined the association link to compose process and business concepts. For example, we have added workspace and document management concerns for our Actol customer [13], and deployment and dynamic selection concerns for our Schneider customer [14].

Non functional concerns are added through the introduction of annotations (similar to stereotypes in UML) on the process model elements. For example, we have added, using this technology, the security concern for our Thales customer [15].

FOCAS extension also provides ways to ground activities to a large range of possible actors. Natively, APEL grounds activities to humans (agendas and TODO lists); it remains the default if no other grounding is defined. The system has

provisions for grounding toward any piece of executable code, being legacy, COTS, component of any component model, local or not [16].

4 FOCAS Engineering Environment

In order to build a complete service-based application following the above approach, the developer has to define a model for each concern, how these models are related (association, grounding, binding etc.) and to generate or create the different engineering artifact. We believe (and we have experimented) that the resulting complexity is overwhelming and cannot be undertaken manually.

For us, the real use of the separation of concerns paradigm, in all domains, and in service-based application in particular, is hampered by the lack of support. FOCAS goal is to provide a comprehensive environment that makes multi-concerns of service-based applications easy to design and develop.

To that purpose, FOCAS provides a specialized editor for relevant models (e.g. an APEL editor, and a data editor), and relies on the Eclipse platform and existing tools for traditional models (Java editor for Abstract Service definition and Behavior model development). More important, FOCAS consistently manages the interoperability between these tools, representations and models and provides Wizards for concerns composition. This section presents shortly the FOCAS environment.

4.1 Composing Business Process and Concepts: the Codele Wizard

The business process expresses its logic only based on the knowledge of the business concept and of what they are intended for, irrespective of their actual name, content and representation. We have developed the Codele tool (0) which purpose is to help in defining the meta-link between concerns (at metamodel level), and links at model level. This tool was used to define all our concern compositions, for example, composing the business process and data structure concerns was defined as a meta-link between metatypes *ProductType* (in process metamodel) and *Data* (in data metamodel), and composing models as a link between a product type name defined in the business process model with a data type defined in data model. In our example the product type *Client* defined in the process model corresponds to the *Customer* type in concept model. The other mappings for our example are shown in the screenshot of the 0.

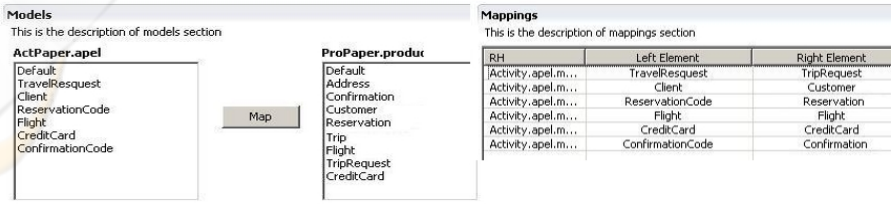

Fig. 3. Composing Business Process and Data.

4.2 Grounding, Binding and Mediation Wizards

Grounding consists in associating an activity (in the business process model) and a service operation (in the abstract services model). The FOCAS grounding wizard, based on information found in the activity and some heuristics, presents those abstract services operations that could be an implementation of that activity. 0 presents an association between the *DoReservation* activity and the reservation operation of the *AirlineService* service.

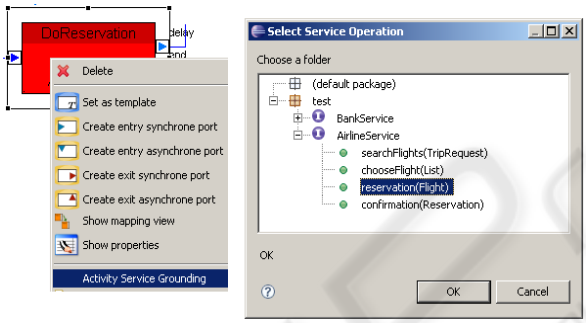

Fig. 4. Activity – Abstract Service Grounding.

The Binding wizard performs in a similar way that Grounding wizard. Provided an abstract service (defined by a Java interface), and a set of actual web services (defined in WSDL files), it selects the possible candidate web services and ask the developer to choose the right one (if more than one is found). FOCAS generates the code required to transform the grounding toward the abstract services into a call toward the actual selected web services.

If no real service is found, the process mediation wizard asks the user if there is a place to look for a convenient service, or if a sub-process definition has to be defined to implement the abstract service as a low-level orchestration of real services.

The data mediation wizard helps in developing the data mediator between the business concepts representation and the one used in the actual web services (WSDL). A XML/Java binding framework is used to transform XML Schema data description into Java classes, therefore in our environment, a mediator is a Java method that takes as input a data of a given type (e.g. real service XML/Java type transformation) and returns the “same” data of another type (e.g. the business concept Java interface representation), and vice versa. Mediation tools can also used, among them we can mention tools developed in the semantic-web community by our partners: X2O mappings for translating from XML to Ontologies [17] and WSMO (Web Service Modeling Ontology) [4] for performing ontology transformations.

Once the binding performed, FOCAS automatically includes the needed mediators before and after calling the real (web) services. The person who defines the binding is not aware that mediation may take place because the binding tool only shows those services that either directly or indirectly (through an available mediator), fit the abstract service.

5 Implementation and Validation

The FOCAS environment is provided as an Eclipse feature. The concerns presented above are technically represented as Eclipse IDE artefacts: projects, folders and files. FOCAS users interact only with models editors and wizards, not directly with the Eclipse artefacts. When a new service-based application is created, existing processes, business concepts and service models can be selected from their corresponding repositories and associated to the new application. The selected models, their associated files and directories are copied in the right place and transformed into the corresponding IDE concepts. In practice, a service-based application is mapped to an eclipse project, where each model is contained in a number of files in a directory, and each wizard generates a composition file containing associations performed between concerns and metadata information for code generation.

Each wizard is configured with heuristics such that, in many cases, the wizard can perform the job with limited human interaction. In well-known business contexts, only the business process model is required from designers, everything else is deduced from the context and service repository, allowing business experts, who are usually not software experts, to easily define advanced (non-trivial) service-based applications.

Any change in a model or composition definition automatically triggers the rebuilding of the whole application. From the FOCAS user point of view, a unique orchestration compiler generates all the application executable code.

FOCAS emphasizes a top-down approach (defining first the business process), which is seldom feasible in WS-BPEL, and concern extension, which is simply impossible in WS-BPEL; comparisons can only be made using a bottom-up approach (starting from existing real services). The effort required developing the first implementation in FOCAS (with a business process being a low-level orchestration) and WS-BPEL is similar. But as soon as maintenance and evolution is to be taken into account, things are different. Suppose that in our toy example, we want to use another bank; it likely involves changing (slightly) the *Customer* concept to *Client*. In FOCAS, the binding goes toward the new bank and a simple mediator (*Customer* to *Client*) is added. In WS-BPEL, we have to declare a new variable of the type *Client* and make the synchronization of data instances *Customer* and *Client* using Assign activities (if the transformation is very simple, calling a specific web service otherwise) before and after each web services call. The invocation activities must be changed to use the new Partner Link (new Web service), new input and output variables and new operation name.

6 Related Works

Several initiatives investigated web services composition; most of works focus on process-based composition (i.e. orchestration). In [18], composition is split into a core process, described in WS-BPEL, and a set of business rules implemented either in AO4BPEL or a business rules engine. The principal advantage presented in [18] is

that rules can evolve independently of the process model, improving reusability. We can consider business rules as a subset of our behavioral model.

The WS-BPEL low abstraction level of is one of its man drawbacks. In [19], this problem is solved by modeling process using UML activity diagram formalism. Then a set of rules is specified to transform this model into a WS-BPEL business process implementation, allowing bidirectional reconciliation of evolving models. In [20] a tool is proposed to check the properties of a process model design against its WS-BPEL implementation. Scenarios are modeled in UML, in the form of Message Sequence Charts (MSC) and implementations are created in WS-BPEL, then design and implementations are translated into Finite State Process (FSP) and finally implementation is validated. In our approach the process model is created using a high level abstract formalism: APEL which is not transformed but directly interpreted.

In WSMO [4] and OWL-S [3] formal semantic languages (WSML and OWL respectively) are proposed for describing semantic relevant aspects of web services, to facilitate its discovery, selection, composition and invocation. In WSMO, ontologies are the basis in this approach and they are used to express the used terminology in services. Mediators between ontologies are used to solve the data incompatibility problem. In our approach, service discovery and selection is not addressed, we rely in the existence of these components provided by our partners in the SEEMP project [21]. To solve data incompatibility problems we use mediation mechanism but ontological mediation could be used instead.

7 Conclusions

Services in general and web services in particular have made much progress toward better independence between parts (no dependency between services, dynamic discovery), and technology independence (using neutral standards: XML and SOAP). From that point of view, services represent the ultimate evolution in the search of reuse. Therefore composing services should become the privileged way to build new applications; it is what orchestration is meant for. Unfortunately, the actual orchestration technology is too low-level and primitive to fit the needs: no abstraction mechanism, no computing capabilities, rudimentary control primitives and so on. Furthermore, emphasizing the reuse dimension means that clients and providers of service do not necessarily match, lexically (names), syntactically (types) and semantically (functions). In the general case, mediation is required.

The current research trend consists in developing a number of independent technologies to address different technical issues (ontology, alignment, choreography interface and so on), and in addressing different functional and non functional concerns separately. In practice, building a new application, reusing existing services, and “weaving” different concerns requires such a lot of heterogeneous technologies and such a high level of expertise that it turns out to be as complex, if not more, than before.

FOCAS innovates in proposing a single and consistent framework where business experts can define the application they need only defining or reusing high level

models. Then, FOCAS assists the developer in “grounding” the high level application definition toward the actual available services (or any other available piece of code), automatically inserting mediation when needed and transparently generating the executable code.

The resulting framework and its user interface are simple to understand and use. To reach that result, FOCAS hides the technology, it consistently reuse (sic) state of the art technologies developed in different fields: services, of course, but also AOP, mediation, and most of all, model and metamodel composition. None of these technologies is visible from the user perspective. FOCAS is seen as an Eclipse plug with a set of editors and wizards.

We believe that FOCAS represents the current answer to the basic Software Engineering principles: information hiding (encapsulation, abstraction) and reuse (decomposition, part library, and composition). High level models are the new technology for encapsulation and abstraction. Reuse relies on the availability, on the web, of very many services; decomposition is performed in term of abstract services and a number of concerns; composition is performed based on model and metamodel composition (conceptually), Aspect Oriented Programming (AOP; technically) and mediation (process and data) to solve the various incompatibilities.

We consider that the major contribution of our work is the consistent use of these recent technologies at the service of the old and fundamental basic software engineering principles, inside a single, consistent and simple to use framework

CADSE and FOCAS are available at http://cadse.imag.fr

References

1. OSGi Alliance.: OSGi 4.0 release. Specification available at http://www.osgi.org/ (October 2005)
2. Alonso, G., Casati, F., Kuno, H., Machiraju, H.: Web Services - Concepts, Architectures and Applications. Springer Verlag (2003)
3. OWL Services Coalition.: OWL-S 1.1 release. Specification available at http://www.daml.org/services/owl-s/1.1/ (November 2004)
4. Roman, D., Keller, U., Lausen, H., Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. Applied Ontology 1(1) (2005) 77–106
5. Thatte, S.: XLANG: web services for business process design. Microsoft, Specification available at http://www.gotdotnet.com/team/xmlwsspecs/xlang-c/default.htm (June 2001)
6. Leymann, F.: Web Service Flow Language (WSFL 1.0). IBM, Specification available at http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf (May 2001)
7. OASIS: Web Services Business Process Execution Language. Specification available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf (april 2007)
8. Khalaf, A., Mukhi, N.K., Weerawarana, S.: Service-oriented composition in bpel4ws. In: Proceedings of the 12th International World Wide Web Conference, Budapest (May 2003) 1–10
9. Blow, M., Goland, Y., Kloppmann, M., Leymann, F., Pfau, G., Roller, D., Rowley, M.: BPELJ: BPEL for Java. A Joint White Paper by BEA and IBM (March 2004)
10. Kloppmann, M., Koening, D., Leymann, F., Pfau, G., Rickayzen, A., von Riegen, C., Schmidt, P., Trickovic, I.: WS-BPEL extension for people - BPEL4People. A Joint White Paper by IBM and SAP (July 2005)

11. Estublier, J., Dami, S., Amiour, M.: APEL: A graphical yet executable formalism for process modeling. Automated Software Engineering: An International Journal 5(1) (January 1998) 61–96
12. Kaiser, G.E., Popovich, S.S., Ben-shaul, I.Z.: A bi-level language for software process modeling. In: In 15th International Conference on Software Engineering, John Wiley & Sons (1993) 132–143
13. Estublier, J., Sanlaville, S.: Extensible process support environment for web services orchestration. International Journal of Web Services Practices 1(1) (2005) 30–35
14. Marin, C., Lalanda, P.: Docosoc - domain configurable service-oriented computing. In: Proceeding in 5th IEEE International Conference on Services (SCC'07). (July 2007) 52–59
15. Chollet, S., Lalanda, P.: Security specification at process level. In: IEEE International Conference on Services Computing (SCC'08). (July 2008)
16. Estublier, J., Villalobos, J., Le, A.T., Sanlaville, S., Vega, G.: An approach and framework for extensible process support system. 2786 (2003) 46–61
17. Kopecky, J., Roman, D., Fensel, D.: Semantic Web Services Grounding, Guadeloupe, Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services, IEEE Computer Society (19-25 February 2006) 127–132
18. Charfi, A., Mezini, M.: Hybrid web service composition: business processes meet business rules. In: ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing, New York, ACM Press (November 2004) 30–38
19. Koehler, J., Hauser, R., Sendall, S., Wahler, M.: Declarative techniques for model-driven business process integration. IBM Systems Journal 44(1) (January 2005) 47–65
20. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Tool support for Model-Based Engineering of Web Services Composition. Volume 1., Orlando, 2005 IEEE International Conference on Web Services (ICWS 2005), IEEE Computer Society (11-15 July 2005) 95–102
21. Della Valle, E., Cerizza, D., Celino, I., Estublier, J., Vega, G., Kerrigan, M., Ramirez, J., Villazon, B., Guarrera, P., Zhao, G., Monteleone, G.: SEEMP: an Semantic Interoperability Infrastructure for e-government services in the employment sector, Innsbruck, Proceedings on European Semantic Web Conference, Springer (3-7 June 2007) 220–234