

A Distributed Aspect Model for Composite Service

Kumaravel Ganesan

Research Scholar
Dept. of Computer Science
Dr. M.G.R University
Chennai, India
kumaravel@ieee.org

Swarup Kumar Mohalik

Senior Researcher
India Science Lab
General Motors TCI
Bangalore, India
swarup.mohalik@gmail.com

Cyril Raj

Professor and Head
Dept. of Computer Science
Dr. M.G.R University
Chennai, India
cyrilraj@hotmail.com

Abstract

Multiple component web services can be composed together to generate a complex service. Creating and deploying such composite web services has lot of practical challenges. One of them is the issue of non-functional service behaviors which cross-cut child component services. They should be separated out from the core-functionality of service instances and need to be handled based on the functionality of the composite service and the component services. The separation makes modular design and deployment easy. However, due to the distributed nature of the services, composite service instances may not be able to access the component service states without infrastructure support. In this paper we propose a novel solution which supplements the web services infrastructure by a distributed aspect infrastructure (DAI). It provides a specification language to separately design the non-functional service concerns as distributed aspects. It also provides a runtime to automatically deploy and execute the aspects, while the web service infrastructure concerns itself with the core-functionality of the composite service. We have implemented a prototype of DAI runtime using SmartFrog and AspectWerkz.

Keywords Service oriented Architecture, Web Service Composition, Aspect Oriented programming. Distributed Aspect Model.

1. Introduction

Web services encapsulate and expose a set of functionalities and state information, available at a web node through standard interfaces (14). This interface standardization and object oriented nature allows the services to get accessed in platform- and implementation-independent ways by other applications. This also enables interoperability among applications running on distributed and heterogeneous environments. Therefore, the web service platform seems to have great potential for integrating different business applications in a seamless manner (6; 7; 8).

Alongwith the mushrooming growth of web services, there has been a compelling need to combine atomic (or,

component) web services to construct Composite Web Services (9). Cost-effectiveness resulting from reuse and value addition are the major drivers for composite services. For a simple example, consider a *Stock price* service which returns the stock price in a single currency. However, users in different countries would like to see the stock price in their local currency. A composite service which combines *Stock price* service and an *Exchange Rate* service with the profile of a user can serve the international customer in a better way.

Typically a web service implementation realizes many non-functional features along with its own core functionality. SLA adaptation, web service policy, security, transaction and business semantics are such non-functional features (10). More often than not, a non-functional behaviour is conjoined with other services functionalities. For example, an authentication procedure may be necessary for each endpoint interface in a service. Because these types of non-functional features cut across many places in a service implementation, they are often referred to as *cross-cutting concerns*.

A composite web service imposes some non-functional features on the component services. This non-functional features need to be adapted in one or more services which are used to build the composite service (13). For example, the 'all-or-none' transaction semantics may require the cancellation service (functional feature) provided by a Toy Store service if an offered gift coupon could not be availed from a partner Book Store service due to failure of authentication (non-functional feature) of a loyalty program. From this point of view, we say that the non-functional features of composite services cross-cut the participating component web services.

The requirements from composite web services can be anticipated and integrated in the component web service interfaces, but this solution is not scalable due to the open nature of web services. Also the solution has high maintenance cost as non-functional features are cross-cutting and have to be updated across all component services if any cross-cutting concern changes. Hence, it is desirable to design the composite web services independently and let the compo-

nent web services adapt themselves to the demands from the composite service dynamically.

Traditionally, Aspect Oriented Programming paradigm has been used to address the problem of adaptation for cross-cutting concerns in various scenarios with a plethora of tools and methods supporting the paradigm (1; 3). In this paper, we extend the paradigm to the distributed case and provide a framework to support the modular development and deployment of composite web services. The proposed solution supplements the existing web services infrastructure by a distributed aspect infrastructure (DAI). It provides a specification language to separately design the non-functional service concerns as distributed aspects. It also provides a runtime to automatically deploy and execute the aspects, while the web service infrastructure concerns itself with the core-functionality of the composite service. We have implemented a prototype of DAI runtime using SmartFrog (4) and AspectWerkz (15).

In the context of web services, AOP has been a natural choice to address several problems (12; 11). AWED (5) is one dynamic aspect model which supports distributed nature of services. But it doesn't support a high-level description of join points based on system states. As a result, specification of cross-cutting concerns of composite services is difficult in this model. The state based join point model suggested by (16) is developed for non-distributed AOP. Also advice weaving pattern has not been explored and discussed in the paper. Our model is different from the existing solutions in several ways: first, it introduces a new state-based join point model for distributed systems; second, the model of the cross-cutting functionality is succinctly captured via transition systems and third, the framework allows dynamic deployment of the distributed aspects using an open source framework of SmartFrog. While the methodology is applicable to a very general case of composite web services, in this paper we restrict ourselves to non-hierarchical compositions for the ease of exposition.

The rest of the paper is organized as follows. In Section 2, we describe composite web services in the Service Oriented Architecture for web services and introduce the problem we want to solve. Section 3 describes the AOP paradigm and the need for a distributed framework to address the problem. In Section 4, we propose the solution framework and follow it up with a case study in Section 5. We conclude with summary and possible future directions in Section 6.

2. Service Oriented Architecture (SOA)

The objective of Service Oriented Architecture is to provide mechanisms to bridge heterogeneous platforms allowing data to flow across various platforms. Using this architecture, software entities which provide a set of functionalities can be made available over the internet as services. These SOA based web services expose description of services in a publicly accessible registry; a service client dis-

covers those services by querying the registry and binds to the selected service dynamically. This dynamic binding capability results in looser coupling between applications. It also helps applications to efficiently adapt to a changing environment.

2.1 Service

A software service can be defined as a software entity which encapsulates functionality and state that can be used by other services. It can also be described as a *state transition system* where the state represents current configuration of the service. The transitions correspond to update operations to these configurations. State information is usually exposed by services so that other services can access and manipulate the exposed states.

2.2 Composite Service

Independent simple services can be composed to achieve more complex functionality as a service. This *composite service* can be specified by using languages like BPEL4WS and executed by runtime engines such as BPEL4J. Typically, a composite service is orchestrated in a *centralized* fashion through a master node. This master node is responsible for coordination of control flow and data exchange between individual services. Alternatively, one can have a *decentralized implementation* comprising multiple runtime engines. Each runtime engine executes a part of composite service specification at a remote location and communicate with other engines directly.

2.3 Separation of Cross-cutting Concerns in Composite Services

In a composite service environment, there are many non-service behaviors which crosscut across multiple component service implementations. Example of these distributed cross-cutting concerns could be Interoperability, Security and Transaction Management, etc. In order to have a modular implementation of the atomic component services, and composite services built using these components, the cross-cutting concerns need to be separated out from the component service implementation. Further, we need to have better mechanisms to specify the concerns. As an example of composite service, we consider a travel service (See Figure 1) constituted by two component services: Train Reservation Service (TRS) and Hotel Booking Service (HBS).

Each component service may have its own transactional behavior which is generally based on notions of composable, atomic and non-atomic tasks. The service interface has to expose various operations to support this transactional behavior. Based on the service interface details, for example, one can conclude that it supports either short lived atomic transaction or long running business activity transaction. The support for particular transaction type also includes adding more restriction in the service implementation like predefining time duration of blocking resources or allow only rele-

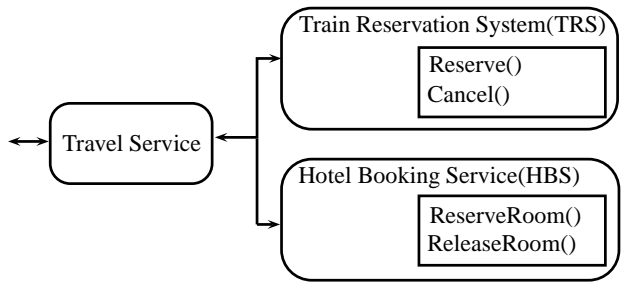


Figure 1. Example Scenario.

vant user to call a compensation service. In another scenario, there is a need for adaptation of services based on the QOS requirement of the composite service. Hence these QOS related behavior changes should be considered as cross-cutting concerns and separated out from all the component web services.

In our example, HBS may support pivot type transaction, which means it doesn't support any rollback mechanism. TRS may be a long running service and it has its own internal state like wait listed availability, RAC (reservation against cancellation) availability and confirmed availability. Based on the state of the TRS service, if ticket is not available then the compensation functionality (releaseRoom()) in HBS service should be enabled for the particular user. Similarly, if hotel booking is not available then the compensation functionality (Cancel()) in TRS service should be enabled.

3. Aspect Oriented Program (AOP)

A software entity can be considered as a combined implementation of multiple concerns. Concerns are goals or objectives of any given piece of code or software. Some concerns of a given software module focus on the business logic, or in other words, the core concern of the module. There could be several other concerns for the same module that focuses on issues such as persistence, security, logging, performance monitoring, tracing, etc. Typically, the latter-mentioned concerns, which could also be referred to as cross-cutting concerns, are implemented across many different modules.

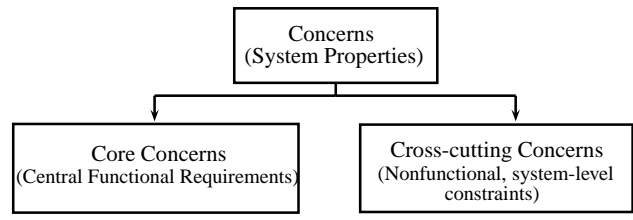


Figure 2. A Software Entity.

AOP (1) is an extension to (and not a replacement of) the traditional programming techniques such as procedural, structural or Object-oriented programming (OOP). It allows the software designer to separate the non functional concerns from the core functional concerns and helps to imple-

ment them as a separate module. This *separation of concern* provides better modularity in the software system. AOP introduces a notion of *aspect*, an entity to specify the cross-cutting concerns. OOP or any other traditional programming technique can be used to develop the core function concerns as a baseline system. Implemented aspects are weaved on the core module to produce the final executable code. A typical aspect unit will have two parts as *Join point* and *advice*. A join point specifies an execution point in the program and an advice is the set of instruction which gets executed at the join point.

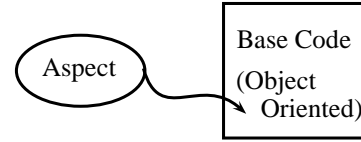


Figure 3. Aspect Oriented Programming.

Dynamic AOP (2) allows runtime-weaving mechanism, which enables one to weave aspect code snippets into the application during runtime. Concurrent Aspect (17) provides the possibility of weaving multiple advices in a single joint point.

4. Proposed Aspect Model for Composite Web Service

The existing AOP paradigm provides a simple fixed code based join point model. But in a composite service environment, we need a notion of distributed, dynamic aspect where advices are weaved/unweaved at join points of component web services. Further, the weaving/unweaving should happen when specified sequences of events occur at component service hosts participating in the composite web service. We propose a Distributed Aspect Infrastructure (DAI) which, along with the web service infrastructure (e.g. BPEL), provides a flexible solution for design and development of composite web services and the cross-cutting concerns (See Figure 4 for a schematic of the relation between component web services, DAI and composite web services).

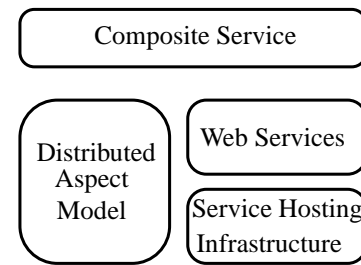


Figure 4. Components of DAI.

DAI is a combination of core service composition description (CSD) in BPEL and cross-cutting concern description (CCD) in a distributed aspect model (DAM). Similar

to the web service runtime engine BPEL4J, DAI defines a runtime engine for DAM supported by local aspects and a distributed deployment framework.

4.1 Distributed Aspect Model (DAM)

As mentioned briefly above, a distributed aspect that models a cross-cutting concern must specify the following:

1. The set of events from different component service hosts that are necessary to infer the global states (or a part of it) relevant to the concern
2. Specific sequences of the global states which determine the dynamic join points, and
3. Distributed advices to be weaved based on the dynamic join points

Accordingly, the distributed aspect model (DAM) is designed as a triple $\text{AspectModel} = \langle \text{MonitorSpec}, \text{BehaviourSpec}, \text{AdviceSpec} \rangle$. The following Figure (Figure 5) shows the three components of DAM. Note that the MonitorSpec and BehaviourSpec together determine a distributed join point and AdviceSpec defines the distributed advice. In the following, we explore each of the components of DAM in more detail.

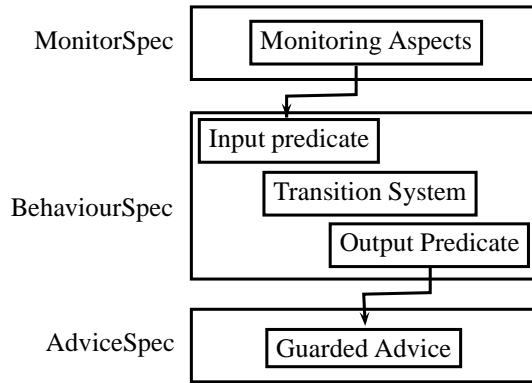


Figure 5. Distributed Aspect Model.

4.1.1 Monitor Spec

MonitorSpec of a distributed aspect is specified as a set of triples of form $\langle \text{event_name}, \text{predicate}, \text{service_name} \rangle$. Each triple is deployed at the component service host through a dynamically weaved local aspect of the host. The pointcut of these local system aspects is the execution point where the specified predicate changes its value. The advice part of this local aspect invokes the notification mechanism of DAI runtime to notify the BehaviourSpec.

4.1.2 BehaviourSpec

BehaviourSpec is specified by a State Transition System (STS) which is the core of the proposed aspect model. The STS has three components: $\langle S, T, O \rangle$, where

- S is a finite set of states, used to capture the global behavior of various web services.
- T is a set of transitions i.e. triples of form $\langle \text{source state}, \text{event}, \text{target state} \rangle$. Here, the events are the notified events from MonitorSpec which capture the state of component web services.
- $O : S \rightarrow \text{Pred}$ is a mapping from the states of STS to a finite set of boolean predicates (Pred). These output predicates are hooks for AdviceSpec to control system execution as shown in the subsequent section.

4.1.3 AdviceSpec

AdviceSpec is a variant of "Distributed Advice"[] and is specified as a triple $\langle \text{guard}, \text{point-cut}, \text{advice} \rangle$, where

- guard is a boolean combination of predicates from Pred,
- point-cut is a pair of $\langle \text{service_name}, \text{local-point-cut} \rangle$, and
- advice is a set of instructions as in a local aspect.

4.2 DAI Runtime

DAI runtime has two parts: a simple non-distributed AOP system to monitor the local events and a distributed deployment framework which supports notification and remote deployment of Aspects. The runtime parses the specified CCD file and weaves local aspects in relevant nodes. Join point of the local advice are the execution points where the interesting events occur. The advice of the local aspect triggers the notification of event on the join point match. The runtime engine then makes the transition in the BehaviourSpec depending upon the notified event. Finally, depending upon the output of the current state of the behaviourSpec, it weaves the guarded advice which is again a local aspect in a particular node.

5. Case Study

Let us consider the example in Figure 1 discussed in section 2.3. Recall that the component services TRS and HBS are composed and exposed as a single composite named as Travel Service, and our goal is to separate the cross-cutting concern of transaction from the component services in the context of Travel Service. In our framework, Travel Service will be represented as a combination of a service composition description (CSD) using BPEL and a cross-cutting concern description (CCD) using DAM. The BPEL part expresses the core service comprising both (1) TRS.Reserve() and (2) HBS.ReserveRoom() in the standard format. In the following, we describe the DAM part in detail.

There are four events of interest in TRS as shown in Figure 6(a) and explained in the following.

1. The status of reserved ticket is changed to Wait Listed (E_twl)
2. The status of reserved ticket is changed to RAC (E_trac)

3. The status of reserved ticket is changed to Confirmed (E_tc)
4. The status of reserved ticket is changed to *Not Available* (E_tna)

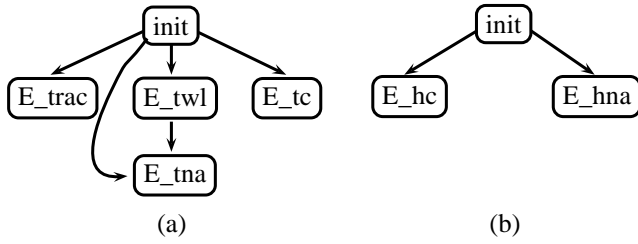


Figure 6. Event model for TRS(a) and HBS(b).

Note that the event E_twl may be followed by E_tna, implying that a wait-listed ticket may not be confirmed. Also, there are two events in HBS as shown in Figure 6(b) and explained below.

1. The status of the Room Booking is changed to *Confirmed* (E_hc)
2. The status of the Room Booking is changed to *Not Available* (E_hna)

The MonitorSpec of the distributed aspect CCD is the set of triples $\{\langle \text{TRS}, P_{\text{twl}}, E_{\text{twl}} \rangle, \langle \text{TRS}, P_{\text{trac}}, E_{\text{trac}} \rangle, \langle \text{TRS}, P_{\text{tc}}, E_{\text{tc}} \rangle, \langle \text{TRS}, P_{\text{tna}}, E_{\text{tna}} \rangle, \langle \text{HBS}, P_{\text{hc}}, E_{\text{hc}} \rangle, \langle \text{HBS}, P_{\text{hna}}, E_{\text{hna}} \rangle\}$. The predicate P_twl become true when the status of reservation is changed to Wait Listed. Other predicates are similar.

The BehaviourSpec of CCD, which is a state transition system is shown diagrammatically in the Figure 7 . It has a default initial state (-, -). The states depict the history of transaction state as derived from the notified event. For example, the state (S-hc, -) says that the event E_hc has occurred in HBS implying the hotel booking is confirmed but there is no information about ticket reservation from TRS. We are particularly interested in sequence of events where one of train reservation or hotel booking was not available. From the transition system it is clear that this sequence of events lead to either state (S-hc, S-tna) or (S-hn, S-tc). The state (S-hn, S-tc) says that the events E_hn and E_tc have occurred, implying hotel booking was not available but ticket reservation was made. In this case, we want to ensure the transaction property "all or none" by enabling the compensation interface in TRS. Similarly, the state (S-hc, S-tna) says that the events E_hc and E_tna have occurred, implying hotel booking is confirmed but train reservation is not made. In this case, we want to enable the compensation interface in HBS.

Note that the output function assigns the value True to the predicate P_hbs in the state $\langle \text{S-hc}, \text{S-tna} \rangle$ and assigns the value True to the predicate P_trs in the state $\langle \text{S-hn}, \text{S-tc} \rangle$. Finally, the AdviceSpec is specified as $\{\langle P_{\text{hbs}}, \text{pc}_{\text{hbs}}, \text{ad}_{\text{hbs}} \rangle, \langle P_{\text{tr}}, \text{pc}_{\text{tr}}, \text{ad}_{\text{tr}} \rangle\}$,

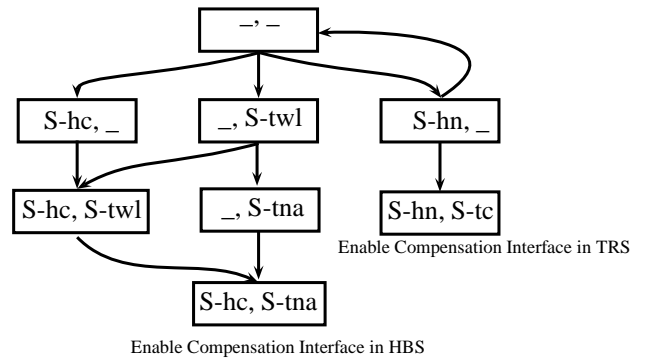


Figure 7. Global State Transition System.

$\langle P_{\text{tr}}, \text{pc}_{\text{tr}}, \text{ad}_{\text{tr}} \rangle\}$, where pc_hbs is a point-cut in HBS, and pc_trs is a point-cut in TRS. The advice ad_hbs (resp. ad_trs) enables the compensation service CancelRoom() (resp. CancelTicket()) for this particular user. This completes the specification of CCD. Our implementation of the proposed aspect model is developed using SmartFrog tool and Aspectwerkz. Here Smartfrog language is used to define DAM and Aspectwerkz along with smartfrog's deployment engine is used to implement the DAI runtime. The runtime behavior of DAM is shown in Figure 8.

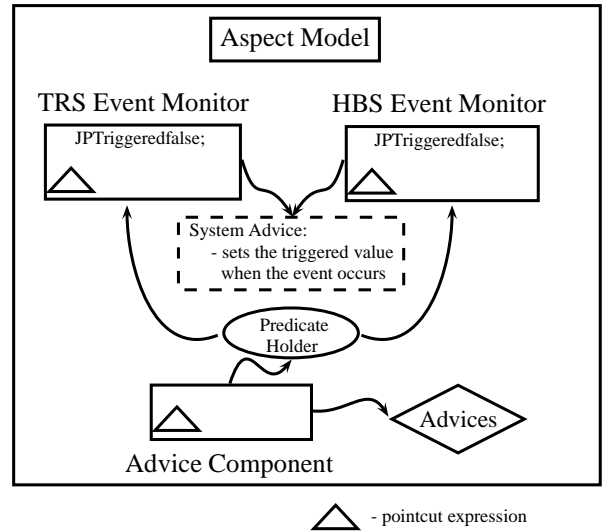


Figure 8. Runtime Behavior of DAM.

6. Summary and Discussion

In this paper, we have proposed a solution for the separation of non-functional features (cross-cutting concerns) of composite web services. The solution extends the existing web services infrastructure to a distributed aspect infrastructure(DAI). We have used SmartFrog as a distributed deployment and event notification mechanism. One can use any other mechanism which supports these functionalities. The

state-transition system of BehaviourSpec in DAI also can be implemented in any programming language.

Our approach provides a rich framework for cross-cutting concern design because of the state-based join point model. Another important benefit of this approach is that the deployment location of the advising component can be determined at runtime. We are now exploring several threads for further optimization and extension of the model.

In order to explore the scalability of the solution, we need to estimate the overhead introduced by DAM and propose optimizations to reduce the overhead. We need to deploy extra queuing mechanisms when multiple advices are prescribed by the BehaviourSpec for a node. In a large system with many events, the bottleneck of BehaviourSpec needs to be avoided by distributing the implementation of BehaviourSpec.

Since BehaviourSpec captures event sequences of a distributed system and the events are asynchronous (no global clock), there may be race conditions leading to different advices output by the BehaviourSpec for the same set of causally unrelated events. In order to avoid this, the transition system of BehaviourSpec should be designed such that it is not able to distinguish between event interleavings i.e. different interleavings must result in same set of actions (weaving of aspects).

Another issue is the real-time aspect of the framework. In order to deploy composite services with real-time constraints among distributed events, we have to extend BehaviourSpec with clocks. We are now exploring these extended features of cross-cutting functions and their modeling in BehaviourSpec.

Acknowledgments

We thank the anonymous referees for their critical comments. It has enabled us to improve the presentation of the paper and reinforced our ideas for future work.

References

- [1] Kiczales, Gregor et al., *Aspect-Oriented Programming*, In Proc. 11th European Conf. Object-Oriented Programming (ECOOP'1997), LNCS 1241, Springer, 1997, pages 220–242.
- [2] Popovici, A., Gross, T., Alonso, G., *Dynamic Weaving for Aspect Oriented Programming*, In Proc. 1st International Conference on Aspect-Oriented Software Development (AOSD'2002), Enschede, The Netherlands, 2002, pages 141–147.
- [3] JAsCo project, <http://ssel.vub.ac.be/jasco/>
- [4] SmartFrog project, <http://www.smartfrog.org>
- [5] Navarro, L. D., Südholt, M., Vanderperren, W., De Fraine, B., and Suvé, D., *Explicitly distributed AOP using AWED*. In Proc. 5th international Conference on Aspect-Oriented Software Development (AOSD'2006), Bonn, Germany, March 2006, pages 51–62.
- [6] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. *Unraveling the Web Services web: An introduction to SOAP, WSDL, and UDDI*. IEEE Internet Computing, 6(2), Mar/Apr 2002, pages 86–93.
- [7] Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M., *A Foundational Vision for E-Service*, In Proc. Workshop on Web Services, e-Business, and the Semantic Web (WES2003) held in conjunction with The 15th Conference On Advanced Information Systems Engineering (CAiSE2003), Klagenfurt/Velden, Austria, 2003.
- [8] K. P. Birman. *Like It or Not, Web Services Are Distributed Objects*. Communications of the ACM, 47(12), December 2004, pages 60–62.
- [9] A. Charfi and M. Mezini. *Hybrid Web Service Composition: Business Processes meets Business Rules*. In Proc. 2nd International Conference on Service Oriented Computing (ICSOC'2004), New-York, USA, 2004, pages 30–38.
- [10] G. Ortiz, J. Hernández, and P. J. Clemente. *Decoupling Non-Functional Properties in Web Services: An Aspect-Oriented Approach*, In Proc. The 2nd European Workshop on Web Services and Object Orientation (EOOWS'2004) held in conjunction with the 18th European Conference on Object-Oriented Programming (ECOOP'2004), Norway.
- [11] Mostefaoui, G. K., Maamar, Z., Narendra, N. C., and Sattanathan, S., *Decoupling Security Concerns in Web Services Using Aspects*, Third International Conference on Information Technology: New Generations (ITNG'2006), Las Vegas, USA, April 2006, pages 20–27.
- [12] Cámara, J., Canal, C., Cubo, J., and Murillo, J. M., *An Aspect-Oriented Adaptation Framework for Dynamic Component Evolution*, Electron. Notes Theor. Comput. Sci., 189, July 2007, pages 21–34.
- [13] Ponnalagu, K., Narendra, N.C., Krishnamurthy, J. and Ramkumar, R., *Aspect-oriented Approach for Non-functional Adaptation of Composite Web Services*, IEEE International Conference on Services Computing - Workshops (SCW 2007), Salt Lake City, USA, July 2007, pages 284–291.
- [14] Singhal, Sharad., Pruyne, James., Machiraju, Vijay., *Picasso: A Service Oriented Architecture for Model-based Automation*, HP Labs Technical Report, <http://www.hpl.hp.com/techreports/2007/HPL-2007-50R1.html>
- [15] Aspectwerkz project, <http://aspectwerkz.codehaus.org/>
- [16] Noorazeen Mohd Ali and Awais Rashid. *A State-based Join Point Model for AOP*, In Proc. 1st ECOOP Workshop on Views, Aspects and Role (VAR05), Glasgow, Scotland, July 2005.
- [17] Douence, Rémi., Botlan, Didier Le ., Noyé, Jacques and Südholt, Mario., *Concurrent aspects*, In Proc. 5th international conference on Generative programming and component engineering (GPCE), Oregon, USA, 2006, pages 79–88.