A Formal Orchestration Model for Dynamically Adaptable Services with COWS

Jorge Fox
Department of Telematics
Norwegian University of Science and Technology
Trondheim, Norway
jfox@item.ntnu.no

***Abstract*—The growing complexity of software systems, as well as, changing conditions in their operating environment demand systems that are more flexible, adaptable and dependable. In many domains, adaptations may occur dynamically and in real time. In addition, services from heterogeneous, possibly unknown sources may be used. This motivates a need to ensure the correct behaviour of the adapted systems, and its continuing compliance to time bounds and other Quality of Service properties. The complexity of Dynamic Adaptation is significant, but currently not well understood or formally specified. This paper elaborates a well-founded model of dynamic adaptation, introducing formalisms written using the process algebra COWS. The model provides the foundation for exploring dynamic adaptation and assessing it against predefined specifications. We consider it a contribution for the design of new models and methodologies for system adaptability.**

***Keywords*-Dynamic Adaptation; Formal Languages; Formal Methods.**

I. INTRODUCTION

Modern software systems typically operate in dynamic environments and are required to deal with changing operational conditions, while remaining compliant with the contracted level of service. The execution context of modern distributed systems environments is not static but fluctuates dynamically, and to provide the expected functional service with the desired qualities, systems need to be adaptable. Software service adaptation supports modification of existing services or inclusion of new ones, in response to inputs from the operating environment. Inputs or triggers for adaptation include changes in the running environment and availability of new services.

When dynamic adaptation (DA) occurs at runtime, it has to be performed within given time bounds, and the resulting system must comply with the execution time established for the system.

Moreover, the resulting system must comply with the execution time established for the system as a whole. Each one of these requirements has to be addressed accordingly. For instance, preservation of data integrity requires a mechanism to verify that the integrity of the data is kept during and after the adaptation. Moreover, timeliness and performance related requirements can be addressed by maintaining the execution times within the desired time bounds.

While the possibility of dynamic service deployment and evolution offers an exciting range of application development opportunities, it also poses a number of complex engineering challenges. These challenges include detecting adaptation triggers, facilitating timely, dynamic service composition, predicting the temporal behaviour of unplanned service assemblies and preventing adverse feature interactions following dynamic service composition. Therefore, a dynamically adaptable service has to be able to identify triggers for adaptation, select and direct an adaptation strategy, and preserve desired QoS properties, while avoiding undesired behaviours during or after the adaptation.

A promising approach to achieving the required properties in Dynamically Adaptable (DA) services is the use of formal methods and techniques, which have been successfully applied for managing complexity and system development to ensure implementations of high quality. Formal methods provide the foundation to verify and validate the behaviour of adaptive programs, as well as, the architecture engineering processes that validate a program against desired functional properties. However, most previous efforts to validate dynamically adaptable services against desired functional properties impose high-verification costs as each adaptation must be separately modelled and verified, making these approaches extremely expensive (See Section II). The result is that building DA services is either expensive or limited to predefined adaptation scenarios. Even more, these approaches focus their analysis on the resulting program neglecting the analysis of the adaptation mechanism itself, that is, the Adaptation Manager (AM). We consider the adaptation mechanism to be of capital importance in the development of DA services, since it is responsible for monitoring compliance of adaptation to desired properties and functionalities. Consequently, this work endeavours to provide a formal model of the AM.

Current approaches to DA offer various mechanisms for handling adaptation, such as Generic Interceptors [1], DA with Aspect-Orientation [2], Dynamic Reconfiguration [3], Dynamic Linking of Components [4], and Model-Driven Development of DA Software [5]. However, a more generic framework to verify the adaptation mechanism against commonly accepted service properties is still needed.

Formal languages provide the underpinnings to describe and model software systems in a precise manner, and are fundamental for the level of analysis, validation and proof required for assuring adaptation compliance to specifica-

 ISBN: 978-1-61208-156-4

tions. In this paper, we propose a concrete model for the AM, which provides the groundwork for a conceptual model. Moreover, the mathematical foundation selected for this study facilitates its validation against predefined QoS properties, such as, availability and responsiveness.

This article is organized as follows: Section II discusses related work. Later on, Section III provides an overview of the language chosen to describe the model. Section IV introduces our formal model and Section V discusses its properties.

Finally, in Section VI we draw some conclusions and introduce future work.

II. State of the Art

We introduce related work in two groups. First, those approaches specifically related to DA, second, formal approaches that are applied to analyse similar families of problems as the ones presented here.

Dynamic Adaptation

Mckinley and Geihs ([6], [7] and [8]) present an overview of DA and its constituents, however they do not advance a formal model or proposal to explore DA, which is the aims of our work. Similarly to the elements of DA we identified, Segarra and André describe a similar model to ours with components that can be customized for different applications, a component in their framework can be provided with a controller which performs the adaptation depending on execution conditions [9]. In our proposal we define one controller, the adaptation manager, that gathers information from supporting services such as timing and execution evaluation in order to perform adaptations. Work has also been carried out to map BPEL to Process Algebras as Ferrara [10], to Pi-calculus as Abouzaid [11], and to Petri Nets as Ouyang et al. [12].

Formal approaches to DA

The work of Laneve and Zavattaro [13] on web services advances an extension to the π-calculus with a transaction construct, the calculus webπ. This model supports time and asynchrony. However it remains at a more abstract level and is not applied to dynamic adaptation. Ferrara [10] relies on process algebra to design and verify web services, this work also allows to verify temporal logic properties as well as behavioural equivalence between services. Compared to this work, our attempt is more general and is directed at the study of dynamic adaptation. Finally our proposal is aimed at identifying a formal service-oriented language for modelling dynamic adaptation, rather than advancing techniques for formal verification of web services or services as in the work of Ferrara. Mori and Kita [14] explore the use of genetic algorithms to dynamic environments and offer a survey on problems of adaptation to dynamic environments. The work of ter Beek at al. [15] reviews service composition approaches with respect to a selection of service composition characteristics and helps to underscore the value of formal methods for service analysis at design, specially service composition. The authors present a valuable analysis of formal approaches to service composition and elaborate a useful comparison. Amano and Watanabe [16] explore mechanisms to check consistency.

This work ventures at providing a concrete model of an AM on top of which a more abstract one can be built. Furthermore, the streamlined relation between Cows and the model checker CMC permits us to realize a formal analysis of the concrete model we introduce in Section IV.

III. The Service-oriented Language COWS

As a first step towards developing a model of dynamic adaptation, we explored a number of languages in [17]. Current service-oriented formal languages like PiDuce [18], SOCK/JOLIE[19], COWS[20], KLAIM [21], and SCC [22] offer each a different range of possibilities to model DA. Each of these languages has an underlying process algebra and constructs that support definition of services and expressing substitution and deactivation processes. The main method in PiDuce to model service substitution is through virtual machines while, in the case of COWS, it is modelled by delimited receiving and killing activities handled with its process calculus. JOLIE and PiDuce offer no deactivation process. After reviewing these languages, we concluded that the best fit language for modeling the runtime dynamic adaptation problem is COWS. Comparing the characteristics of the languages selected only COWS provides constructs for timing analysis. Timeliness was not the only deciding criteria, considering that0 adaptation of services only via channel renaming is sufficient to achieve DA is questionable, as is the case of PiDUce. In this regards, the composition mechanisms of SOCK/JOLIE are more adequate, yet again with no possibility to evaluate timeliness. The choice is clear and well founded.

The calculus for orchestration of web services (COWS) [23] is a new process calculus similar to the Business Process Execution Language (WS-BPEL) [24]. However, contrary to WS-BPEL, COWS provides a formally specified distributed machine. COWS is a process calculus that allows to specify service-oriented applications, as well as, modelling dynamic behaviour. COWS also provides support for the development of tools to check that a given service composition follows desired correctness properties and unexpected behaviours are avoided.

COWS has been extended with timing elements [20] which facilitate adoption of the language for modelling services with timing requirements. Therefore, this language represents an important line of research in service-oriented computing (SOC). The motivation of the authors to develop COWS is that most formalisms “do not model the different aspects of currently available SOC technologies in their completeness” [20].

 ISBN: 978-1-61208-156-4

Services are structured activities built from basic activities, such as the empty, kill, invoke, receive, and wait. Services are composed by means of prefixing, choice, and parallel composition. Constructs as protection, delimitation, and replication are also provided. The language is parameterised by a set of expressions. Partner names and operation names can be combined to designate communication endpoints, written "$p \bullet o$", where p is a partner and o an operation. These represent activities of type receive "$p \bullet o?$" or invoke $p \bullet o!$".

Timed activities are frequently exploited in service-oriented computing and are needed to model time outs. Passing of time is modelled synchronously for services deployed on the same 'service engine' and asynchronously otherwise. In this language, time passes synchronously for all services in parallel, given that services run on the same service engine. An important aspect in dynamic adaptation is represented by timing constraints for either the execution of a new service or the execution of the system as a whole, therefore, elements for modelling time are needed

COWS [25] provides a proxy mechanism to add compensation handling in existing services. This means that it may add behaviour to existing services in the system by appending a compensatory service. A compensatory service is one that adds to the base service and achieves some corrective or compensatory functionality. We can also avail of sequential composition for adapting an existing protocol to the new one.

In COWS basic actions are *durationless* and the passing of time is modelled by explicit actions. Imperative and orchestration constructs support specification of assignment in variables, conditional choice and sequential composition. Conditional choice and sequential composition of services can be used to achieve dynamic adaptation by composing services with existing ones. The language also defines the concept of service engines, where each engine has its own clock which is synchronised with the clock of other parallel engines. All instances of a service run within the same engine. Moreover, time elapses between each evaluation of expressions and these evaluations are instantaneous. Only the time construct $\oplus_{argument}$ consumes time units. Time elapses while waiting for invoke or receive activities and the argument for wait-activities "$\oplus$" is set to the current stand. Parallel composition of engines in this language is given by sequential composition. Moreover, COWS provides a deactivation activity that forces termination of all unprotected parallel activities. Sensitive code can be protected from killing by placing it into a "protection". COWS computational entities are called *services* [23]. Services in COWS do not have interfaces since communication is realised through message passing among services, which are structured activities built from basic activities.

Finally, COWS can model several typical aspects of web services technologies, for instance, multiple start activities, receive conflicts, routing of correlated messages, service instances and interactions among them [20].

IV. A Model for Dynamic Adaptation

In previous work [26], we identify several techniques to achieve DA. Most of the current DA techniques are static, and more importantly, the flexibility of adaptation or the level at which adaptations are achieved, is in most cases limited. An area of opportunity we identified, is the need for a formal approach in DA. A more formal approach to DA allows system designers and architects to perform an analysis of the adaptive elements in the system. This is the motivation behind our work.

This section presents a two-step process for establishing our DA model.

1) A scenario explaining a dynamic adaptation problem
2) Design of the model in a formal language suitable for verification

Scenario

We assume an all-electronic toll system where tolls are collected with the use of a transponder mounted on the windscreen of each vehicle. This is a straightforward case, however, we further assume that the car travels along a number of borders and jurisdictions. In each jurisdiction the toll system may actually differ from the previous one. We propose an adaptable electronic toll collection system on the client (vehicle) side. This system would allow the owner to install only one toll system that interacts with the toll booth at each contact point. In this scenario we assume the car receives a welcoming signal every time it approaches a tollbooth. This is illustrated in Figure 1. In this diagram we restrict the representation to the sequence of events in the interaction between three interacting roles `Toll booth`, `Vechicle Communication`, and `Toll booth ETS service`. As we can see, the Toll booth sends a welcome signal to every vehicle approaching it. On receipt of the signal, the vehicle's communication service requests the type of payment protocol from the toll booth. The Vehicle identifies the protocol and evaluates its compatibility to the one currently installed. In case it is not compatible, the Toll booth is notified and requests a new protocol for the car to be sent together with an estimate of the car's arrival time to the toll collection area. This estimate is used as the upper time bound for the subsequent adaptation of the vehicle's Electronic Toll Payment Service.

In the following we present a model for DA suitable for validation against desirable attributes of services and service-oriented computing applications. According to this, a service should be: Available, Reliable, and Responsive.

Model

The model of DA is introduced in Figure 2. Here we assume a runtime monitor in the software composite that activates the AM based on predefined triggers.

 ISBN: 978-1-61208-156-4

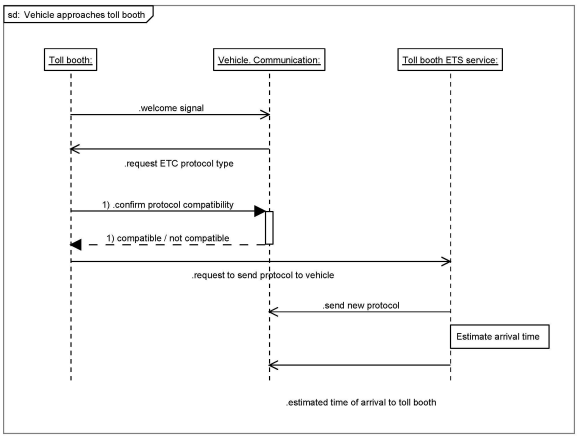


Figure 1. Sequence Diagram Tollbooth

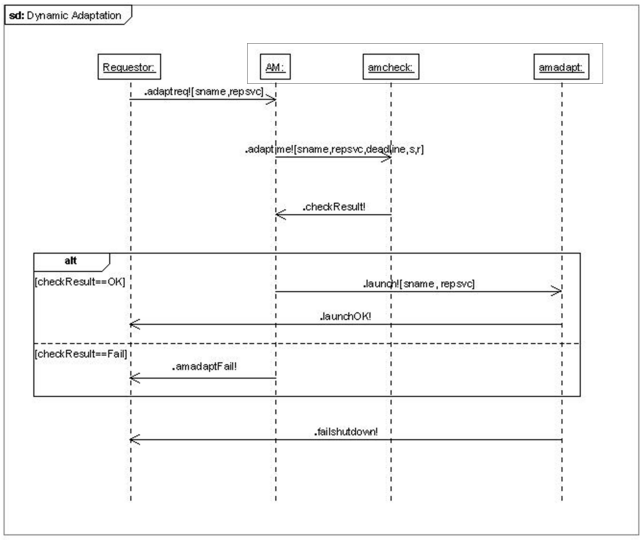


Figure 2. Sequence Diagram Adaptation Process

The adaptation process initiates by sending a signal (`adaptreq(sname,repsvc)!`) from the `Requestor` to the `AM`. This is represented in Listing 1 line 11. The AM further sends the timeliness parameters to evaluate the plausibility of the adaptation in view of the time constraints, as illustrated by the second message `adaptime(sname,repsvc,deadline,s,r)`. Once the object "amcheck" receives this signal, the timeliness conditions are evaluated and the result returned to the AM. The flow for the adaptation proceeds, when the service `amcheck` is called with the time estimations in Listing 1 line 7. In this point, a procedure to compare the estimated time of the adaptation against the predefined upper bound is called as expressed in Listing 2 lines 12 to 18. Next, in case the adaptation can be fulfilled within the defined time bounds, a second condition is evaluated. This is represented

```
1
let
adaptManager(service) =                                          3
    * [X][Y][Z][XX][YY]
    service.create?<X,Y,Z,XX>.p.adaptime!<X,Y,Z,XX>              5
  | [X][Y][Z][XX]
    ser.checkOK?<X,Y,Z,XX>.q.exectime!<Z,XX>                     7
  | ser.checkFail?<>. ser.launchFail!<repsvc>
  | ser.checkFail2?<>. ser.launchFail!<repsvc>                   9
requestor() =
    serv.create!<0,4,10,60>                                      11
  | ser.checkOK2?<>.amadapt.launchOK!<> |
    (amadapt.launchOK?<>.s.signalOK!<>                           13
    + ser.launchFail?<repsvc>.s.signalFail!<>
    + ser.launchFailx?<>.s.signalFail!<>)                        15
in
adaptManager(serv)                                               17
| requestor()
| * amcheck()                                                    19
| amcheck2()
| s.signalFail?<>.nil                                            21
| s.signalOK?<>.nil
end                                                              23
```

Listing 1
ADAPTATION MANAGER, REQUESTOR AND MAIN SERVICE

```
Amcheck_gt_deadline(X) =
  (ser.checkFail!<>)                                             2
Amcheck_le_deadline(X,Y,Z,XX) =
  (ser.checkOK!<X,Y,Z,XX>)                                       4
Amcheck_gt_deadline2(X) =
   (ser.checkFail2!<>)                                           6
  | memory.assert?<X>.nil
Amcheck_le_deadline2(X) =                                        8
  (ser.checkOK2!<>)
amcheck()=                                                       10
    [X][Y][Z][XX]
    p.adaptime?<X,Y,Z,XX>.                                       12
    [i#]
    (i.selectgreater!<X gt Y> |                                  14
      (i.selectgreater?<true>.
       Amcheck_gt_deadline(X) +                                  16
       i.selectgreater?<false>.
       Amcheck_le_deadline(X,Y,Z,XX)                             18
      )
    )                                                            20
```

Listing 2
ADAPTATION-TIME CHECK

in Listing 3 lines 3 to 10. This evaluation verifies that the adaptation preserves the overall execution time of the service. In case both conditions are uphold, the service `amadapt` is executed. After execution, this service sends the signal `cheqOK2!` (Listing 1 line 12) to the requestor, which in turn notifies the user on a successful adaptation by sending the signal `launchOK`. Otherwise, the AM responds `signalFail`.

The model focuses on two timeliness conditions, adaptation time and execution time. However, other conditions can be analyzed by the AM in the same manner. For instance, verifying that the reconfigured service is in a quiescent state before performing any changes in order to avoid interaction and coordination conflicts. In the following section, we describe the verification process that the model grants.

 ISBN: 978-1-61208-156-4

```
amcheck2()=
    [X][Y]
    q.exectime?<X,Y>.
    [i#]
    [K]
    (i.selectgreater!<X gt Y> |
      (i.selectgreater?<true>.
       Amcheck_gt_deadline2(X) +
       i.selectgreater?<false>.
       Amcheck_le_deadline2(X)
      )
     )
```

Listing 3
EXECUTION-TIME CHECK

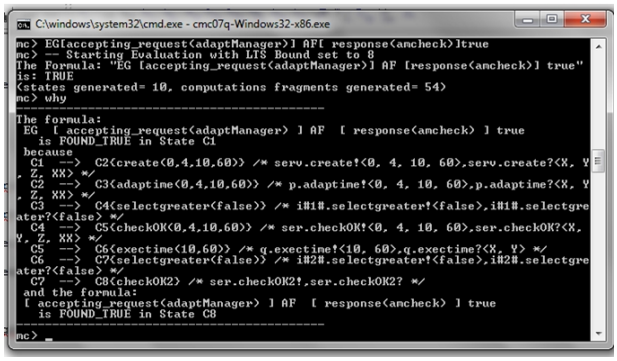
V. ANALYSIS OF THE MODEL

Dynamic software architectures allow us to build dynamically adaptable systems by supporting the modification of a system's architecture at runtime. Possible modifications include structural adaptations that change the system configuration graph, for example, the creation of new and the deletion of existing components and connectors, as well as functional adaptations where components are replaced. Further, even more challenging changes are those that modify the behaviour of components and ultimately services.

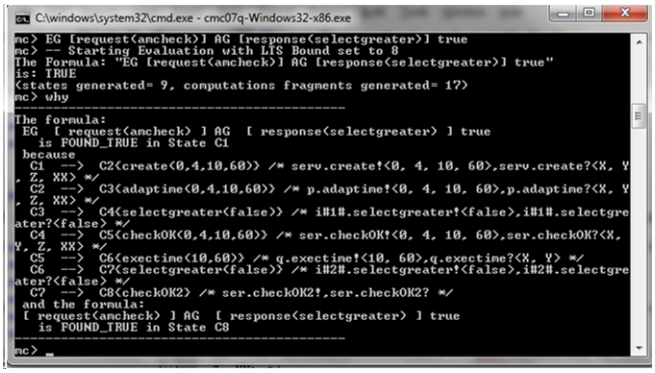
To achieve reliable dynamic adaptable services we need to evaluate service modifications against a number of quality of service properties. Services must comply as a minimum with the following three properties. The first one, Responsiveness, means that the service always guarantees an answer to every received service request, unless the user cancels. The second property, Availability, requires that the service is always capable to accept a request. Finally, the third property, Reliability means that the service request can always succeed.

In order to evaluate a system against the properties mentioned above, we first express these properties and the model in a language that supports a formal analysis. This analysis should be performed preferably in an automated manner. Cows, the language we selected, possesses this quality. In the following we introduce the selected model checker, which is CMC. Cows grants straightforward coupling with CMC. Afterwards, we explore the model for reliability and responsiveness properties with the model checker.

For reasons of space, we present only the model checking results for Responsiveness and Reliability. To explore the first property, Responsiveness, we define a formula with a universal quantifier on the execution of `amcheck` following a call to the AM specified as an existential quantifier to the AM, as shown in Listing 4. A run of the model checker on this condition returns `true` and no counter-example as shown in Figure 3(a). Verification of the model against Availability follows a similar pattern. In general terms, the User identifies the service name of interest, in this case the AM and specifies the related logical condition as a universal quantifier over the request for the adaptation manager. The AM initial call is found in Listing 1, which is input to the model checker and the conditions is tested.



(a) Responsiveness Check with CMC



(b) Reliability Check with CMC

Figure 3. CMC Runs

```
AG[request(adaptManager)]true
EG[accepting_request(adaptManager)] AF[ response(amcheck)] 2
    true
```

Listing 4
RESPONSIVENESS

```
EG[request(requestor)] EG[response(launchOK)] EG[response( 1
    launchFail)]true
```

Listing 5
RELIABILITY

The formula for Reliability is built by the combination of existential quantifiers over the services `requestor`, `launchOK` and `launchFail` in Listing 5. The model-checker renders no counter-example, but `True` as shown in Figure 3(b).

VI. CONCLUSIONS

Real time DA is an area of research that poses new challenges to software development, where the goal is to produce a system capable of adapting to changing conditions in the operational environment. Additional demanding goals related to DA are the integration of new services as these become available, or coping with reconfiguration issues, all this

at runtime and under time constraints. DA has been proposed to provide solutions for these challenges. A methodology for the study of DA is still an open question. Formal methods have been in use for a long time in the computer science community and a number of new approaches and formal languages are available. We suggest that modelling DA with a formal language can provide precise answers to most of the existing questions and grant a better understanding. As a consequence, in this work, we recommend the use of the formal language COWS to model DA, introduced a first formal model, and assessed it by checking against three widely accepted service properties: Responsiveness, Availability and Reliability, with the model checker CMC.

ACKNOWLEDGMENT

This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme

REFERENCES

[1] S. M. Sadjadi and P. K. McKinley, "Act: An adaptive corba template to support unanticipated adaptation," *icdcs*, vol. 00, pp. 74–83, 2004.

[2] Z. Yang, B. H. C. Cheng, R. E. K. Stirewalt, J. Sowell, S. M. Sadjadi, and P. K. McKinley, "An aspect-oriented approach to dynamic adaptation," in *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM, 2002, pp. 85–92.

[3] M.-C. Pellegrini and M. Riveill, "Component management in a dynamic architecture," *J. Supercomput.*, vol. 24, no. 2, pp. 151–159, 2003.

[4] C. Escoffier and R. S. Hall, "Dynamically adaptable applications with iPOJO service components," in *Software Composition*, ser. Lecture Notes in Computer Science, M. Lumpe and W. Vanderperren, Eds., vol. 4829. Springer, 2007, pp. 113–128.

[5] J. Zhang and B. H. C. Cheng, "Model-based development of dynamically adaptive software," in *ICSE '06: Proceeding of the 28th international conference on Software engineering*. New York, NY, USA: ACM, 2006, pp. 371–380.

[6] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "A taxonomy of compositional adaptation," Dept. Computer Science and Engineering, Michigan State University, Tech. Rep. MSU-CSE-04-17, 2004.

[7] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56–64, 2004.

[8] K. Geihs, "Selbst-adaptive software," *Informatik-Spektrum*, 2007, 0170-6012 (Print) 1432-122X (Online).

[9] M.-T. Segarra and F. André, "A framework for dynamic adaptation in wireless environments," in *TOOLS '00: Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 33)*. Washington, DC, USA: IEEE Computer Society, 2000, p. 336.

[10] A. Ferrara, "Web services: a process algebra approach," in *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*. New York, NY, USA: ACM, 2004, pp. 242–251.

[11] F. Abouzaid, "A mapping from pi-calculus into bpel," in *Proceeding of the 2006 conference on Leading the Web in Concurrent Engineering*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2006, pp. 235–242.

[12] C. Ouyang, E. Verbeek, W. M. P. van der Aalst, S. Breutel, M. Dumas, and A. H. M. ter Hofstede, "Formal semantics and analysis of control flow in ws-bpel," *Sci. Comput. Program.*, vol. 67, no. 2-3, pp. 162–198, 2007.

[13] C. Laneve and G. Zavattaro, "Foundations of web transactions." Springer, 2005, pp. 282–298.

[14] K. H. Mori Naoki, "Genetic algorithms for adaptation to dynamic environments - a survey," in *26th Annual Conference of the IEEE Electronics Society IECON 2000*, I. P. I. E. Conference), Ed., 2000.

[15] A. B. Maurice H. ter Beek and S. Gnesi, "Formal methods for service composition," *Annals of Mathematics, Computing & Teleinformatics*, vol. 1, 5, pp. 1–10, 2007.

[16] W. T. Amano Noriki, "Towards constructing component-based software systems with safe dynamic adaptability," in *International Workshop on Principles of Software Evolution (IWPSE)*, 2001.

[17] J. Fox and S. Clarke, "An analysis of formal languages for dynamic adaptation," in *International Conference on Engineering of Complex Computer Systems (ICECCS 2010)*, March 2010, pp. 3 – 13.

[18] S. Carpineti, C. Laneve, and L. Padovani, "PiDuce – a project for experimenting web services technologies," *Science of Computer Programming*, vol. 74, no. 10, pp. 777 – 811, 2009.

[19] C. Guidi, R. Lucchi, R. Gorrieri, N. Busi, and G. Zavattaro, "SOCK : A calculus for service oriented computing," *Proceedings of Service-Oriented Computing ICSOC 2006*, vol. 4294/2006, pp. 327–338, 2006.

[20] A. Lapadula, R. Pugliese, and F. Tiezzi, "Cows: A timed service-oriented calculus," in *Proc. of 4th International Colloquium on Theoretical Aspects of Computing (ICTAC'07)*, ser. Lecture Notes in Computer Science, vol. 4711. Springer, 2007, pp. 275–290.

[21] L. Bettini, V. Bono, R. D. Nicola, G. Ferrari, D. Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, and B. Venneri, "The KLAIM Project: Theory and Practice," in *Global Computing: Programming Environments, Languages, Security and Analysis of Systems*, ser. LNCS, C. Priami, Ed., no. 2874. Springer, 2003, pp. 88–150.

[22] M. Boreale, R. Bruni, L. Caires, R. D. Nicola, I. Lanese, M. Loreti, F. Martins, U. Montanari, A. Ravara, D. Sangiorgi, V. T. Vasconcelos, and G. Zavattaro, "SCC: A service centered calculus," in *Web Services and Formal Methods*, ser. Lecture Notes in Computer Science, M. Bravetti, M. Núñez, and G. Zavattaro, Eds., vol. 4184. Springer, 2006, pp. 38–57.

[23] A. Lapadula, R. Pugliese, and F. Tiezzi, "A Calculus for Orchestration of Web Services," in *Proc. of 16th European Symposium on Programming (ESOP'07)*, ser. Lecture Notes in Computer Science, vol. 4421. Springer, 2007, pp. 33–47.

[24] "Business Process Execution Language for Web Services (BPEL4WS)," http://xml.coverpages.org/bpel4ws.html, last accessed 01.09.2011.

[25] "Calculus for Orchestration of Web Services (COWS)," http://rap.dsi.unifi.it/cows/, last accessed 01.09.2011.

[26] J. Fox and S. Clarke, "Exploring approaches to dynamic adaptation," in *Proceedings of the 3rd International DiscCoTec Workshop on Middleware-Application Interaction*. New York, NY, USA: ACM, 2009, pp. 19–24.

 ISBN: 978-1-61208-156-4