# An Extensible Monitoring and Adaptation Framework*

Razvan Popescu, Athanasios Staikopoulos, Siobhán Clarke
Firstname.Lastname@cs.tcd.ie

Distributed Systems Group, Trinity College Dublin, Ireland

**Abstract.** Several techniques have been defined for the monitoring and adaptation of applications. However, such techniques usually work in isolation and cannot be easily integrated to tackle complex monitoring and adaptation scenarios. Furthermore, applications may have special requirements which make it difficult to reuse such off-the-shelf approaches. In particular, these requirements may cross several application layers such as – the organisation of stakeholder roles, coordination of planned activities, and integration with third-party services.

In this paper we outline a lightweight, loosely-coupled and extensible monitoring and adaptation framework that allows application developers to integrate monitoring and adaptation techniques as units that can be linked to solve complex requirements and achieve cross-layer adaptation. In order to cater for application-tailored adaptation units, we propose a pattern-based technique for the development and integration of adaptation units.

## 1 Introduction

Complex service-oriented applications are generally heterogeneous, loosely-coupled, long-lived and continuously running and have to cope with frequent changes to their requirements and environment. In order to address such changes, applications need to be inherently flexible and adaptive and supported by appropriate adaptation infrastructures. The adaptation process serves, for example, to ensure that the application is fault tolerant or compatible with new clients. The adaptation can be enforced either at design, or at run-time, and it can be triggered by the human designer or operator of the application, or by a monitoring process.

Several techniques have been defined for the monitoring and adaptation of applications. They generally tackle issues such as interface [5], behavioural [3], quality-of service [7], service-level agreement [8], or policy mismatches [6]. However, such techniques usually work in isolation and cannot be easily integrated to tackle complex monitoring and adaptation scenarios [10]. Furthermore, applications may have special requirements which make it difficult to reuse such off-the-shelf approaches. In particular, these requirements may cross several application layers such as – the organisation of stakeholder roles, coordination of planned activities, and integration with third-party services [2].

In this paper we outline a lightweight, loosely-coupled and extensible monitoring and adaptation framework that allows application developers to integrate monitoring and adaptation techniques as *units* that can be linked to solve complex requirements and achieve cross-layer adaptation. In order to cater for application-tailored adaptation units, we propose a pattern-based technique for the development and integration of adaptation units.

In a nutshell, the framework assumes an *event bus* through which monitoring and adaptation units can communicate following a publish-subscribe mechanism. Human operators and monitoring units trigger events under certain conditions (e.g., a repeated service invocation failure may eventually trigger the replacement of the respective service). These events lead to the execution of matching adaptation units. In order to cope with complex adaptation scenarios, the framework allows developers and integrators to construct complex adaptation units either as compositions

* This work has been supported by the EU Alive project [2].

of existing units, or by linking them through events. We employ workflows for the high-level definition of adaptation-unit behaviour, and we name such workflows *adaptation patterns*. Using these patterns, developers can either integrate existing adaptation techniques, or develop new ones. We propose the use of the Yet Another Workflow Language (YAWL [12]) as language to express adaptation patterns. An important advantage of using YAWL as high-level language is the possibility to define workflows that integrate adaptation tasks that map onto existing adaptation techniques exposed as (WSDL [13]) Web services.

Throughout this paper we shall illustrate the applicability of the framework in the context of the ALIVE project [2]. ALIVE proposes an advanced model-driven engineering framework for the disciplined and systematic development, deployment and management of service-oriented applications based on organisation and coordination mechanisms often seen in human and other societies. ALIVE distinguishes three conceptual levels of design: the *organisation level* (viz., application stakeholder roles and their relationships), the *coordination level* (viz., high-level workflows that coordinate planned activities needed to fulfil application goals) and the *service level* (viz., supporting services that enact workflow tasks). The designs are hierarchical, with highly dependent layers, where concepts in one level refer to or reuse concepts defined in another.

The remaining of this paper is structured as follows. Section 2 presents an ALIVE use case that motivates the need for an extensible, cross-layer monitoring adaptation framework. Section 3 briefly describes the proposed framework. Section 4 illustrates how the framework supports the adaptation of the use case in section 4. Finally, section 5 presents some concluding remarks and future directions of work.

## 2 Crisis Management Use Case: Problem

Figure 1 presents a generic emergency-handling system [2]. The organisation level describes the roles involved in the scenario and their relationships. The main goal (viz., objective) of the EMERGENCY CENTRE is to rescue, assist and transport wounded to hospitals. The emergency centre fulfils this goal by dividing it into several sub-goals and delegating them to the other roles. The emergency centre receives the emergency call and acts as an orchestrator for the other roles. It delegates the rescue operations of the trapped people to the FIRE STATION role, the assistance and transport of wounded to the FIRST-AID STATION role and the traffic management operations to the POLICE STATION role.

The role of the coordination level is to describe the workflows needed to fulfil the application goals. In ALIVE [2], human and software agents fulfil organisational roles and are in charge of executing workflow tasks either manually, or by mapping them onto (Web) services. The coordination level in Figure 1 shows a possible workflow needed to fulfil the goal of assisting the wounded[1]. The workflow starts by assessing the magnitude of the incident. This task is performed through the invocation of a corresponding (Web) service. The workflow continues by computing a route to the location of the incident (again done through the invocation of a (Web) service) and then by instructing the medical team to proceed to the location of the incident. Next, the medical team assists the wounded. The workflow then computes a route to the nearest hospital and instructs the medical team to transport the wounded. Finally, the workflow instructs the medical team to fill in a report which is to be processed through the invocation of a (Web) service. The goal of the first-aid station – the assistance and transport of wounded – can be split into two sub-goals that can be mapped onto the successful execution of particular workflow tasks, that is, *Assist Wounded* and *Proceed to Hospital*, respectively.

In this scenario we employ the workflow in Figure 1 to rescue people affected by a hurricane. We assume that the execution of the workflow proceeds as planned up the transportation of wounded to hospitals. Due to a major flood, the ambulances cannot follow the planned route to the hospital and an alternative route would force them to loose precious time. The medical team informs the emergency centre of this issue and the emergency centre operator decides to trigger an

[1] We have used YAWL [12] to graphically represent the application workflow.

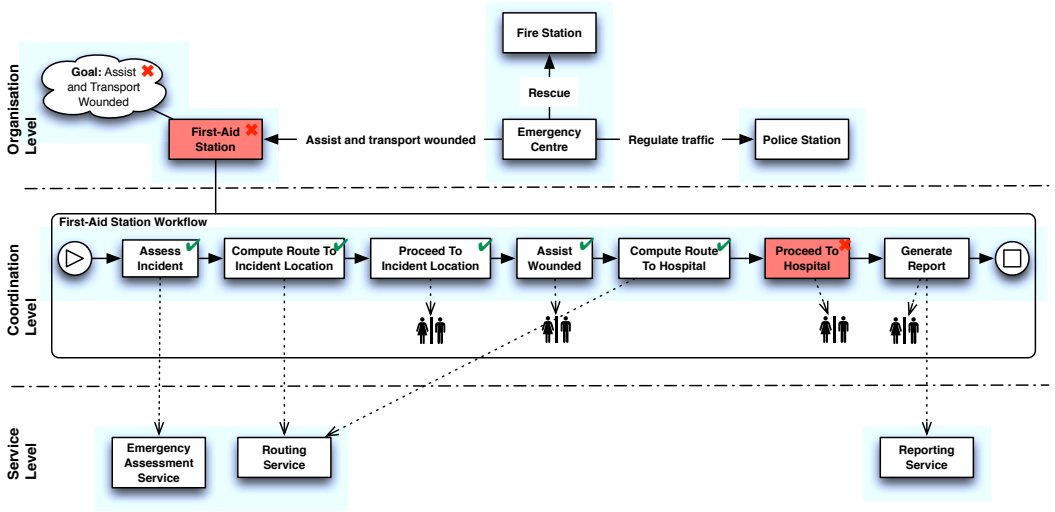


**Fig. 1.** ALIVE Architecture of a Crisis Management Use Case.

alternative plan that involves transporting the wounded using helicopters from a nearby military base.

This issue requires reorganisation and adaptation at various levels of the application. On the one hand, the organisation level needs to introduce a new role – MILITARY BASE – whose goal is to transport the wounded to hospitals. On the other hand, the workflow at the coordination level has to reflect the changes needed to achieve this goal using the newly added role. For example, the new workflow has to find a route to the helicopter base first. These changes can be accommodated through two adaptation techniques – one that resolves organisational issues (e.g., based on [9]), and another one that resolves workflow mismatches (e.g., along the lines of [3]).

In Section 4 we shall describe how the monitoring and adaptation framework copes with such triggers and facilitates dynamic changes at various levels of the application.

## 3 Overview of the Monitoring and Adaptation Framework

Figure 2 presents the high-level architecture of the proposed monitoring and adaptation framework. The architecture employs an event bus (e.g., an Enterprise Service Bus [4]) to separate the core of the application (viz., the computational aspects of the application) from monitoring and adaptation units. The event bus follows a *publish-subscribe* mechanism. Monitoring and adaptation units and human application operators and designers register themselves as publishers of events. Then, other adaptation units and human application operators and designers register themselves as consumers of events by subscribing to the events of their interest.

This architecture brings the following main benefits:

- **Lightweight application architecture.** Applications can be configured to use only the monitoring and adaptation units they need. For example, a video-on-demand application may need a monitoring unit that produces *<Low Download Request>* and *<High Download Request>* events together with an adaptation unit that, upon the reception of such events modifies the upload/download bandwidth of the application.
- **Loosely-coupled monitoring and adaptation.** The event bus act as mediator between application and monitoring and adaptation units, which can be added, removed, or updated without breaking down the interoperability with the application. Furthermore, the proposed architecture allows cross-layer adaption to be performed by linking adaptation units through events (e.g., please see subsection 3.2 and section 4 for details).

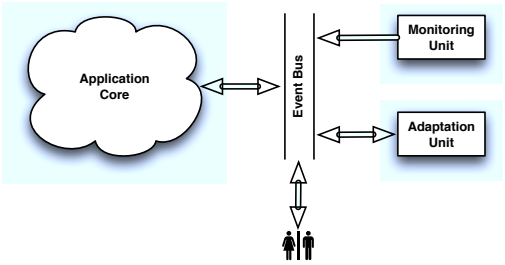


**Fig. 2.** Architecture of the monitoring and adaptation framework.

| Application Level | Triggers | Events |
|---|---|---|
| Service Level. | Service invocation fails repeatedly,<br>Located better service,<br>... | Service Failure,<br>Match and Replace Service,<br>Adapt Service Interface,<br>Adapt Service Behaviour,<br>Warning,<br>Critical Failure,<br>... |
| Coordination Level. | Task execution has failed,<br>... | Task Failure,<br>Replace Task,<br>Generate Workflow and Replace,<br>... |
| Organisation Level. | Objective cannot be achieved,<br>... | Objective Failure,<br>Adapt Organisational Structure,<br>... |
| Designer/Operator. | Requirements have changed,<br>Environment has changed,<br>... | Any of the above. |

**Table 1.** Examples of ALIVE events.

– **Extensible monitoring and adaptation framework.** As applications evolve, new monitoring and adaptation units may be deployed so as to cope with changing application requirements. New and existing monitoring and adaptation techniques can be integrated into the proposed architecture by wrapping them as event producers and/or consumers.

As previously mentioned, we shall illustrate the applicability of the monitoring and adaptation framework in the context of the ALIVE project [2]. The architecture of an ALIVE application consists of three levels: organisation, coordination, and service, as shown for example in Figure 1. Recall that the organisation level describes the application stakeholder roles and their relationships. The coordination level describes high-level workflows needed to fulfil application goals. Finally, the service level describes supporting services that enact workflow tasks. In the next two subsections we present some example ALIVE events and adaptation patterns.

### 3.1 ALIVE Events

Table 1 presents some of the generic events that can be employed in the process of monitoring and adaptation of ALIVE applications. Monitoring and adaptation units can be configured to trigger adaptation events at different application levels given certain conditions are met. For example, a service level monitoring unit may trigger a <*Service Failure*> event when it notices that a service invocation has failed repeatedly. Or, an adaptation unit that tackles the adaptation of the application's workflow may trigger a <*Match and Replace Service*> event when new workflow tasks need to be enacted by services, and a <*Critical Failure*> event if the workflow cannot be adapted to meet an application goal.

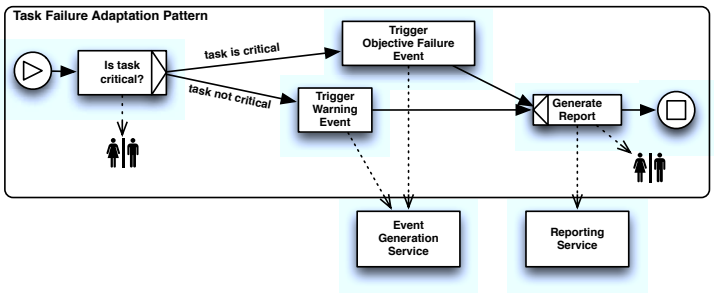


**Fig. 3.** Task Failure adaptation pattern.

### 3.2 Alive Adaptation Patterns

Adaptation units can subscribe to and receive notifications of triggered events (e.g., such as the ones presented in subsection 3.1), as well as can trigger events. Adaptation patterns serve to define the high-level workflow of adaptation units. Such workflows are useful to create complex adaptation patterns from simple ones. For example, an adaptation pattern that is in charge of generating an Alive application workflow so as to meet one of its goals might enclose another adaptation pattern in charge of selecting and adapting services needed to enact the workflow tasks.

As previously mentioned, we argue for the use of YAWL [12] as language to define the workflow of the adaptation patterns. YAWL is a formal language that builds on Petri nets. YAWL conditions and tasks are similar to Petri net places and transitions, respectively. A YAWL workflow specification consists of one or more extended workflow nets (or EWF-nets for short) arranged in a tree-like structure. An EWF-net is a graph where nodes are tasks or conditions, and arrows define the control-flow relation. Each EWF-net has a single input condition and a single output condition. A YAWL task may be either atomic (e.g., *Match Role to Objective* in Figure 4) or composite (e.g., *Adapt Organisational Structure* in Figure 4). An atomic task corresponds to a leaf of the tree. A composite task corresponds to an EWF-net at a lower level in the hierarchy. The EWF-net without any composite tasks referring to it is called top-level workflow and it corresponds to the root of the tree-like hierarchy. Tasks employ one join and one split construct. A join or split control construct may be one of the following: AND, OR, XOR, or EMPTY. For example, *Is Task Critical?* and *Generate Report* in Figure 3 have an XOR split and join, respectively. Tasks can be enacted either by human operators or designers (e.g., *Is Task Critical?* in Figure 3), or by (WSDL) Web services (e.g., *Generate Report* in Figure 3)[2].

Hereafter we present three adaptation patterns that may be used to tackle the issues raised by the use case described in section 2.

Figure 3 presents the YAWL workflow of a possible adaptation pattern that tackles a task failure. The workflow first verifies whether the task that failed (e.g., the *Proceed to Hospital* task in Figure 1) is needed to meet the objective of the application workflow (viz., First-Aid Station workflow in Figure 1)[3]. If the task is critical, the workflow triggers an <*Objective Failure*> event, otherwise it triggers a <*Warning*> event. Finally, the workflow generates a report based on its execution path.

Figure 4 presents the YAWL workflow of a possible adaptation pattern that tackles an objective failure (e.g., in response to an <*Objective Failure*> event triggered by the Task Failure

[2] Although being enacted by services, some tasks may also require inputs from the human operators or designers as well, such as *Generate Report* in Figure 3. Furthermore, note that we did not represent the actors enacting composite tasks since these tasks are usually composed of several tasks, each enacted by a different service or human operator or designer

[3] Please note that we employ YAWL workflows to define both – application workflows and adaptation pattern workflows.

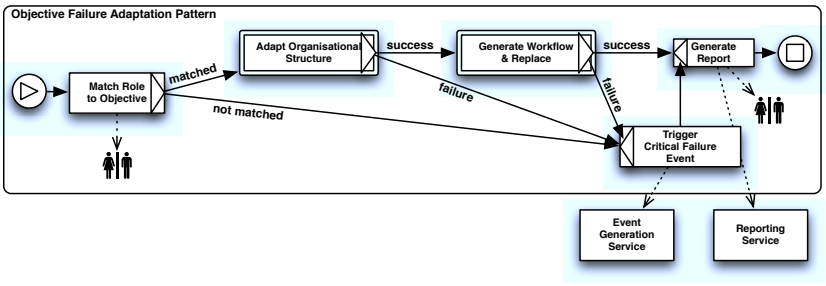


**Fig. 4.** OBJECTIVE FAILURE adaptation pattern.

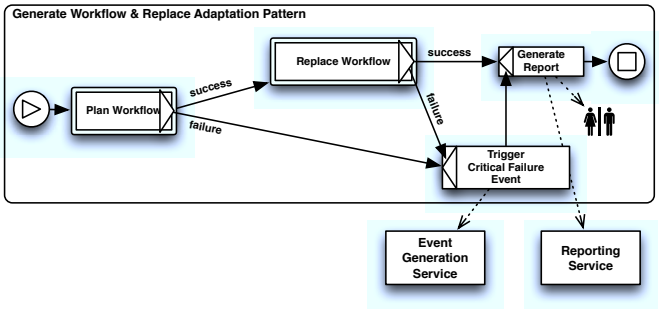


**Fig. 5.** GENERATE WORKFLOW & REPLACE adaptation pattern.

adaptation pattern in Figure 3). The workflow first matches a (possibly new) organisational role that can fulfil the respective objective. Given one exists, the workflow adapts the organisational structure by adding the matched role and by taking into account its relationship with the other roles in the organisational structure, as well as the objective to be fulfilled[4]. A successful adaptation of the organisational structure leads to the definition of a new application workflow that makes use of the added role to meet the objective (see Figure 5). If the generation is successful, the workflow terminates with the generation of a report.

Figure 5 presents the YAWL workflow of a possible adaptation pattern that can be used to generate an application workflow to meet a given objective. The workflow first employs planning and workflow adaptation techniques to generate a workflow that can meet the objective. Next, it replaces a given workflow (e.g., the workflow task *Proceed to Hospital* in Figure 1) with the newly generated workflow. Finally, it generates a report.

In the next section we describe how these three adaptation patterns can be employed to tackle the complex adaptation issues raised by the use case described in section 2.

## 4 Crisis Management Use Case: Solution

For this scenario we assume that the workflow task *Proceed to Hospital* (see the workflow in Figure 1) fails due to flooding.

The monitoring and adaptation framework first triggers a <Task Failure> event which leads to the execution of the TASK FAILURE adaptation pattern, as shown in Figure 3. Since the *Proceed to Hospital* task is critical for the fulfilment of the workflow objective, the adaptation pattern triggers an <Objective Failure> event. This event leads to the execution of the OBJECTIVE FAILURE adaptation pattern. The human operator is in charge of finding a (possibly) alternative organisational role that can fulfil the failed objective. For this scenario we assume that the operator

[4] Please note that space limitations do not allow us to describe the content of all composite tasks of the workflows involved in the use case.

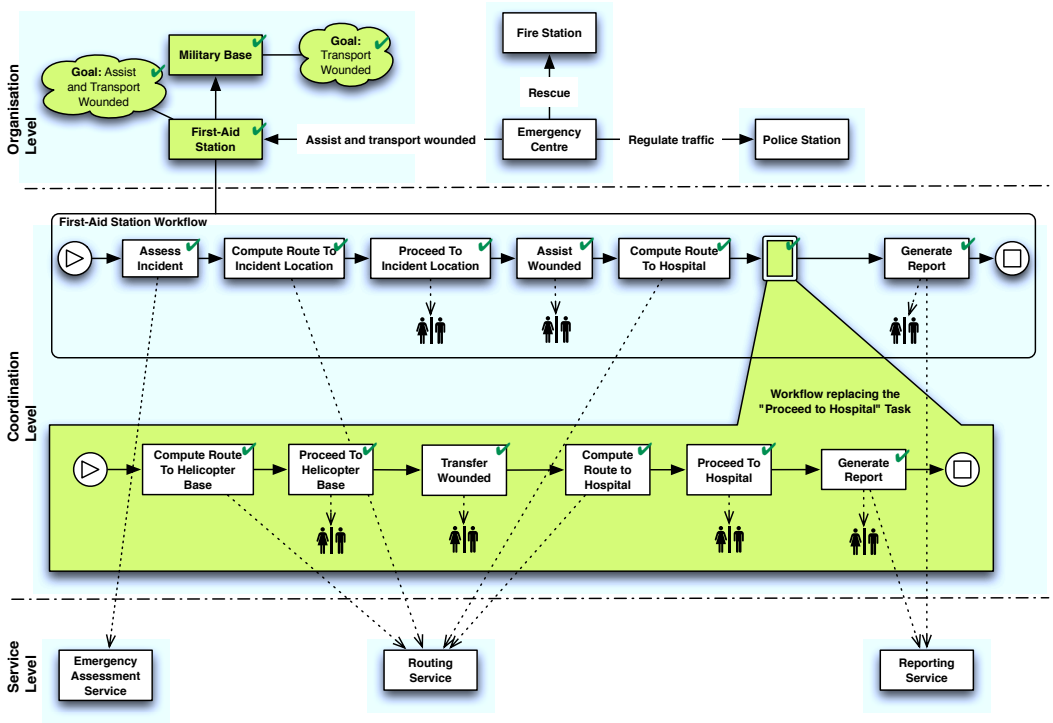


**Fig. 6.** Adapted ALIVE Architecture of the Crisis Management Scenario.

chooses the MILITARY BASE role. The OBJECTIVE FAILURE adaptation pattern continues then with the modification of the organisational structure so as to accommodate the newly matched role. For this process we may assume that the *Adapt Organisational Structure* task of the OBJECTIVE FAILURE adaptation pattern makes use of organisational adaptation techniques based on [9]. Figure 6 illustrates the new organisational structure, in which the FIRST-AID STATION role delegates the transport of the wounded to the MILITARY BASE role.

Next, the adaptation pattern tries to generate a new workflow that takes into account the new organisational structure to meet the (sub-)objective of transporting the wounded to a hospital. This is achieved through the execution of the GENERATE WORKFLOW & REPLACE adaptation pattern, depicted in Figure 5. This pattern first employs human expertise and planning techniques to generate a workflow that can meet the given objective while following the organisational constraints. We may assume that the planning step makes use of workflow adaptation techniques along the lines of [3]. Next, the pattern replaces the failed task with a composite task that encloses the newly created workflow. This can be achieved using YAWL worklets [1].

For this scenario we assume that the planning step produces the workflow in Figure 6. The replacement workflow first routes the medical team to the military base. The medical team then transfers the wounded onto military helicopters. Finally, the workflow computes a route to the nearest hospital and it instructs the medical and military teams to proceed to the respective location.

## 5 Concluding Remarks

In this paper we have outlined a lightweight, loosely-coupled and extensible monitoring and adaptation framework that allows application developers to integrate monitoring and adaptation techniques as units that can be linked to solve complex requirements and achieve cross-layer adaptation. This is supported by an event bus through which monitoring and adaptation units can communicate following a publish-subscribe mechanism.

Developers may either wrap exiting approaches as monitoring and adaptation units, or they may develop new ones. In order to assist developers during this process we have proposed the use of YAWL as language to define the high-level workflow of monitoring and adaptation units. Using YAWL, developers have the possibility to define complex unit workflows as compositions of simple ones, or to integrate existing monitoring and adaptation approaches as (WSDL) Web services enacting workflow tasks. The workflows can be used as patterns and customised into application-tailored monitoring and adaptation units.

As future work, we mainly plan to investigate:

- The development of a monitoring and adaptation framework based on the outline given in this paper,
- The definition of a set of (common,) generic adaptation patterns following the adaptation taxonomy in [11], and
- The use of model-driven engineering techniques for the disciplined and systematic engineering of tools that support the development and deployment of application-tailored adaptation units from generic patterns.

## References


1. M. Adams, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In R. Meersman and Z. Tari, editors, *OTM Conferences (1)*, volume 4275 of *LNCS*, pages 291–308. Springer, 2006.
2. Alive. D2.2a: Theoretical Framework, 2009. Deliverable available from: `http://www.ist-alive.eu/index.php?option=com_docman&task=doc_download&gid=3&Itemid=49`.
3. A. Brogi and R. Popescu. Automated Generation of BPEL Adapters. In A. Dan and W. Lamersdorf, editors, *ICSOC*, volume 4294 of *LNCS*, pages 27–39. Springer, 2006.
4. D. Chappell. *Enterprise Service Bus*. O'Reilly Media, ISBN 978-0596006754, 2004.
5. M. Dumas, M. Spork, and K. Wang. Adapt or Perish: Algebra and Visual Notation for Service Interface Adaptation. In S. Dustdar, J. L. Fiadeiro, and A. P. Sheth, editors, *Business Process Management*, volume 4102 of *LNCS*, pages 65–80. Springer, 2006.
6. A. Erradi, P. Maheshwari, and S. Padmanabhuni. Towards a Policy-Driven Framework for Adaptive Web Services Composition. In *NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices*, pages 33–38. IEEE Computer Society, 2005.
7. J. Harney and P. Doshi. Speeding up Adaptation of Web Service Compositions Using Expiration Times. In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pages 1023–1032, New York, NY, USA, 2007. ACM.
8. N. C. Narendra, K. Ponnalagu, J. Krishnamurthy, and R. Ramkumar. Run-Time Adaptation of Non-functional Properties of Composite Web Services Using Aspect-Oriented Programming. In B. J. Krämer, K.-J. Lin, and P. Narasimhan, editors, *ICSOC*, volume 4749 of *LNCS*, pages 546–557. Springer, 2007.
9. L. Penserini, H. Aldewereld, F. Dignum, and V. Dignum. Adaptivity within an Organizational Development Framework. In *SASO '08: Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 477–478. IEEE Computer Society, 2008.
10. S-Cube. PO-JRA-1.2.1: State of the Art Report, Gap Analysis of Knowledge on Principles, Techniques and Methodologies for Monitoring and Adaptation of SBAs, 2008. Deliverable available from: `http://www.s-cube-network.eu/results/deliverables/wp-jra-1.2/PO-JRA-1.2.1-State-of-the-Art-report-on-principles-techniques-and-methodologies-for-monitoring-and-adaptation.pdf/at_download/file`.
11. S-Cube. CD-JRA-1.2.2: Taxonomy of Adaptation Principles and Mechanisms, 2009. Deliverable available from: `http://www.s-cube-network.eu/results/deliverables/wp-jra-1.2/CD-JRA-1.2.2_Taxonomy_ofAdaptation_Principles_and_Mechanisms.pdf/at_download/file`.
12. W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Inf. Syst.*, 30(4):245–275, 2005.
13. WSDL. Web Service Description Language v1.1, 2001. Available from: `http://www.w3.org/TR/wsdl`.