Towards Ensuring High Availability in Collective Adaptive Systems

David Richard Schäfer[1], Santiago Gómez Sáez[2], Thomas Bach[1], Vasilios Andrikopoulos[2], and Muhammad Adnan Tariq[1]

[1] Institute of Distributed and Parallel Systems (IPVS)
{david.schaefer, thomas.bach, adnan.tariq}@ipvs.uni-stuttgart.de
[2] Institute of Architecture of Application Systems (IAAS)
{gomez-saez, andrikopoulos}@iaas.uni-stuttgart.de
University of Stuttgart, Stuttgart, Germany

Abstract. Collective Adaptive Systems support the interaction and adaptation of virtual and physical entities towards achieving common objectives. For these systems, several challenges at the modeling, provisioning, and execution phases arise. In this position paper, we define the necessary underpinning concepts and identify requirements towards ensuring high availability in such systems. More specifically, based on a scenario from the EU Project ALLOW Ensembles, we identify the necessary requirements and derive an architectural approach that aims at ensuring high availability by combining active workflow replication, service selection, and dynamic compensation techniques.

Keywords: workflows, high availability, service discovery, process fragment injection

1 Introduction

Collective Adaptive Systems (CAS) comprise heterogeneous entities collaborating towards the achievement of their own objectives, and the overall objective of the collective [1]. Such large-scale systems are usually constituted by physical and virtual entities that interact with each other towards achieving individual and collective goals. In the EU ALLOW Ensembles[3] Project we aim to provide support for modeling, executing, and adapting such interactions among entities. We propose to model and manage entities as collections of *cells* encapsulating their functionalities. Entities collaborate with each other to achieve their objectives in the context of *ensembles* describing the interactions among them.

In [1] we proposed the usage of service orchestrations and choreographies to specify the behavior of cells and to define the interactions within ensembles, respectively. However, the achievement of individual and collective goals through interactions among entities raises further challenges for such a system. More specifically, we focus our work on ensuring high availability of cells by means of

[3] EU ALLOW Ensembles: http://www.allow-ensembles.eu/

Published in Proceedings of the First Business Processes in Collective Adaptive Systems: BPCAS'14; Eindhoven, Netherelands, September 8, 2014.

ensuring the continuation of their execution. Moreover, in case of system failures, errors of any type must be handled effectively and efficiently. For this purpose, we present our vision and ongoing work to target such challenges by analyzing and proposing an approach based on workflow replication, service selection and execution, and dynamic service compensation. Existing work ensures high availability in different ways. Parallel service execution focuses on ensuring activity deadlines [2]. However, such approaches increase the overall cost due to the number of accessed services. Logging and checkpointing mask computing node failures but introduce delays during the recovery phase of the system [3]. Another approach is the primary-backup strategy, where a primary node executes the workflow and transfers its state to all backups [4]. Failures of the primary first have to be detected and a new primary has to be selected, which again introduces delays and, thus, unavailability. In previous work, we targeted declarative workflow replication and restructuring to increase availability [5]. In this work, we build on the service-oriented architecture and imperative workflow description for our proposal.

The contributions of this work can be summarized as follows: (1) the identification of requirements based on the EU ALLOW Ensembles motivation scenario and (2) the design of a reference architecture synthesizing workflow replication, service selection, and dynamic compensation techniques to support high availability. The remainder of the paper is structured as follows: starting from a motivation scenario in Sec. 2, we introduce our architectural approach in Sec. 3, and conclude with a summary and future research challenges in Sec. 4.

2 Motivation

To motivate our work, we use the example illustrated in Fig. 1a based on the FlexiBus scenario [1]. Consider employee Adam gets a call from his boss that he needs to attend a meeting on the other end of the city in half an hour. To reach his destination, the workflow depicted in Fig. 1a needs to be executed on a device capable of executing workflows. In the following, we call such devices *nodes*. Consider that the workflow is executed on Adam's mobile device. Mobile environments in specific but any environment in general might endure node, service, and communication link failures. In best case, these failures lead to temporal unavailability. In the worst case, the execution fails. We address this by executing multiple workflow instances on different nodes (*replicated workflow execution*) and invoking several services for the execution of one activity (*service selection and execution*). In general, replication is highly desirable for increasing availability as mentioned by Schaefer et al. [5] and depicted by the replicated execution of Adam's workflow in Fig. 1b. The service needed by the first activity is only available on the internet. When the mobile phone loses connectivity, it does not receive the reply from the service and cannot proceed. However, the replica on the server in the internet continues execution. If the replicas reconnect and synchronize before the execution reaches the last activity, which has to be executed locally on the phone, the availability is then never affected.

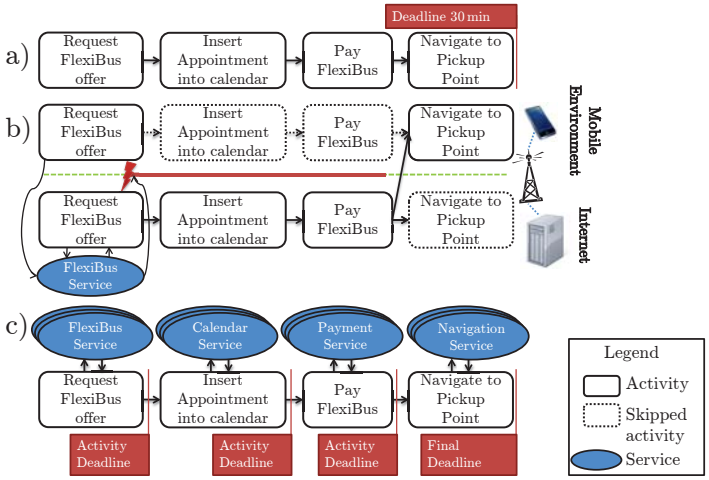

Fig. 1: a) The workflow of Adam to get to the appointment in time. b) Example of a replicated execution. c) Workflow with activity level deadlines and multiple services.

As shown in Fig. 1c, we assume that every activity of the workflow can be satisfied by a number of different services. To guarantee that the workflow deadline is met, we invoke several services for one activity staggered over time. This means we give services time to reply before calling additional services. Both, the workflow replication and the service execution strategy, can lead to the invocation of multiple services which might change the outcome of the workflow execution. In order to compensate for the additional service executions, we propose the dynamic and automated discovery and injection of compensation handlers into the workflow instance during run-time.

3 Approach

In this section, we explain how to use *replicated workflow execution*, and *service selection and execution* to increase availability. Furthermore, our strategies dynamically discover services at run-time making it hard to specify the respective compensation operations at design time. To solve this, we present a dynamic compensation mechanism. Our proposed architecture (cf. Fig. 2) consists of a cluster of *Execution Engines* (EE) (where the replicated workflow instances are executed) each placed on a different node and an *Enterprise Service Bus* (ESB). We add a *Replication Layer* to the EE and the ESB to handle workflow replication. To enable service selection and execution, we add a *Service Selection and Execution Layer* (SEL) to multiple components of the ESB. The components of these layers will be discussed in detail in the remainder of this section.

3.1 Workflow Replication

We use replicated workflow executions to increase availability (cf. Sec. 2). Executing a workflow multiple times on different EEs, placed on different computing

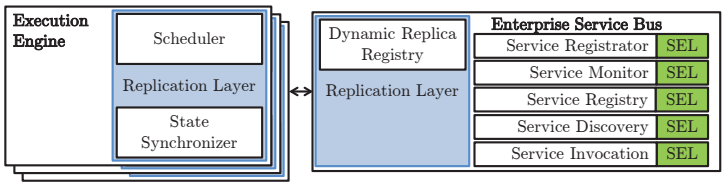

Fig. 2: Our architecture extends the EE and the ESB by a Replication Layer and adds a Service Selection and Execution Layer (SEL) to multiple components of the ESB.

nodes, can lead to inconsistencies. Thus, we appropriately schedule the activities and synchronize the processes, e.g., the payment activity, confirming the offer and transferring the money, is executed only by one EE. Otherwise, Adams pays multiple times. In principle, when executing an activity multiple times, it either can cause inconsistencies (*non-replicable*) or never causes inconsistencies (*replicable*). If the activity is non-replicable, the *Scheduler* (cf. Fig. 2) coordinates to ensure exactly one execution of the activity avoiding any inconsistency in advance. After the activity's execution, the *State Synchronizer* transfers the changed state to all other EEs. To enable seamless communication, all EEs executing workflow replicas are registered in the *Dynamic Replica Registry* (cf. Fig. 2).

To increase availability further, we also allow temporal inconsistencies. Therefore, we further divide the non-replicable activities into compensable and non-compensable. In the case that the EEs cannot synchronize (e.g., because of network partitioning), each EE can proceed executing compensable and replicable activities. When the EEs reconnect, all but one EE have to compensate all non-replicable activities to regain consistency. The Scheduler will coordinate and trigger the necessary compensations. This strategy, however, leads to a new problem. We cannot define the compensation handlers at design time because the compensation logic depends on the used service, which is dynamically discovered during run-time. Thus, our idea is to inject the compensation logic into the workflow instance at run-time (cf. Sec. 3.3).

3.2 Selection and Execution of Services

To guarantee the workflow level deadline, we derive optimal sub-deadlines for all activities (cf. Fig. 1c). To make sure these activity deadlines are met, we investigate the response time characteristics and costs of all services registered in the Service Registrator of the ESB (cf. Fig. 2). For each activity, we invoke a number of services, staggered over time, such that the deadline is met with high probability (i.e., 99.99%) and the expected execution cost (e.g. monetary or energy) is minimized. If an activity is non-replicable but several services are invoked for execution, we need to compensate the additionally executed services dynamically during run-time (cf. Sec. 3.3). This is reflected in additional compensation cost. Currently we investigate approaches of assigning activity level deadlines and invoking services respectively. Preliminary results of a staggered execution, calculated with simulated annealing, promise to decrease the expected execution cost by $\sim 30\%$, compared to invoking all services at the same time.

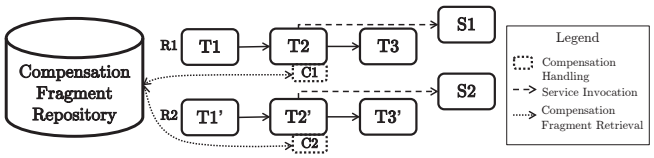

Fig. 3: Retrieval and Injection of Process Compensation Fragments

We plan to extend the existing components of the ESB by a SEL (cf. Fig. 2). For example, the Service Monitor records the response times of the different services and stores that data in the extended Service Registry.

3.3 Dynamic Compensation Handling

The workflow technology natively supports compensation capabilities and proposes the usage of compensation spheres as a mechanism to group transactional and non-transactional activities into a new unit of work [6]. When a compensation sphere is executed unsuccessfully, the workflow system executes the compensation logic defined in the process model for the grouped unit of work. However, two main disadvantages arise when specifying the compensation logic at design time: 1) the process modeler must completely know the external services rollback operations and 2) there exists limited flexibility to dynamically cope with services' compensation logic modifications (e.g., due to service versioning).

The replication and service discovery mechanisms previously presented require dynamic and adaptable compensation capabilities to rollback operations from different services (or service replicas) which are dynamically discovered during run-time. Fig. 3 depicts the proposed approach for enabling an adaptive orchestration of compensation operations based on enabling run-time process refinement and the usage of process fragments presented in [7]. Process fragments partially or completely specify a concrete functionality and are typically persisted and discovered from fragment repositories during the process design time. Process fragments can be used during run-time to refine a process model partially specified during modeling time.

In Fig. 3, two process replicas R1 and R2 discover two services S1 and S2, respectively, which provide similar functionalities. As described in Sec. 3.1, all replicas might execute their discovered services. However, if the activity is non-replicable and compensable, only one execution is selected eventually. This selection leads to the need for compensating state modifications that activities, e.g., T2', perform on the external services. For instance, the compensation logic for a credit card payment varies depending on the payment gateway and credit card issuer. The proposed approach considers the usage of the *Abstract Activity* proposed in [8] to specify compensation fragments placeholders which are refined during run-time. The *Compensation Fragment Repository* enables the storage and retrieval of compensation fragments for different services and their corresponding versions. Therefore, the Replication Layer in both the EE and ESB (cf. Fig. 2) must be aware of such constructs and interact with the compensation fragment repository.

4 Conclusions and Future Work

Collective Adaptive Systems are characterized by the interactions among multiple heterogeneous entities towards achieving individual and collective goals. A system capable of supporting such interactions at the different phases, i.e., modeling, provisioning, and execution, must guarantee high availability and robustness of the functionalities consumed and exposed by such entities. In this work we present our vision towards ensuring such characteristics based on the FlexiBus motivation scenario from the EU ALLOW Ensembles Project.

Based on the requirements identified in the motivating scenario, we present an architectural approach which relies on the usage of workflows and SOA-based approaches and technologies. Moreover, we propose the usage of workflow replication, and service selection and execution strategies towards fulfilling the deadlines which may be imposed by the individual entities. However, such techniques infer the necessity of ensuring an adaptable compensation functionality for the multiple workflow replicas and consumed services, which can be addressed by enabling the dynamic retrieval and injection of compensation fragments. In future work, we plan to identify concrete solutions that can be used for realizing the presented concepts.

Acknowledgement This work has been partially funded by the EU Project ALLOW Ensembles (600792).

References

1. Andrikopoulos, V., Bucchiarone, A., Sáez, S.G., Karastoyanova, D., Mezzina, C.A.: Towards Modeling and Execution of Collective Adaptive Systems. In: Proceedings of WESOA'13, Springer (December 2013) 1–12
2. Stein, S., Payne, T.R., Jennings, N.R.: Robust execution of service workflows using redundancy and advance reservations. Services Computing, IEEE Transactions on **4**(2) (April 2011) 125–139
3. Elnozahy, E.N.M., Alvisi, L., Wang, Y.M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys **34**(3) (September 2002) 375–408
4. Lau, J., Lung, L.C., da Fraga, J., Santos Veronese, G.: Designing fault tolerant web services using bpel. In: Computer and Information Science, 2008. ICIS 08. Seventh IEEE/ACIS International Conference on. (May 2008) 618–623
5. Schäfer, D.R., Bach, T., Tariq, M.A., Rothermel, K.: Increasing availability of workflows executing in a pervasive environment. In: Proceedings of IEEE SCC'2014, IEEE Computer Society (June 2014)
6. Leymann, F., Roller, D.: Production workflow: concepts and techniques. (2000)
7. Eberle, H., Unger, T., Leymann, F.: Process fragments. In: OTM Conferences (1). (2009) 398–405
8. Bialy, L.: Dynamic Process Fragment Injection in a Service Orchestration Engine. Diploma Thesis No. 3564, University of Stuttgart, Germany (2014)