Towards Modeling and Execution of Collective Adaptive Systems

Vasilios Andrikopoulos[1], Antonio Bucchiarone[2], Santiago Gómez Sáez[1], Dimka Karastoyanova[1], and Claudio Antares Mezzina[2]

[1] IAAS, University of Stuttgart
Universitaetsstr. 38, 70569 Stuttgart, Germany
{andrikopoulos,karastoyanova,gomez-saez}@iaas.uni-stuttgart.de
[2] Fondazione Bruno Kessler, Via Sommarive, 18, Trento, Italy
{bucchiarone,mezzina}@fbk.eu

Abstract. Collective Adaptive Systems comprise large numbers of heterogeneous entities that can join and leave the system at any time depending on their own objectives. In the scope of pervasive computing, both physical and virtual entities may exist, e.g., buses and their passengers using mobile devices, as well as city-wide traffic coordination systems. In this paper we introduce a novel conceptual framework that enables Collective Adaptive Systems based on well-founded and widely accepted paradigms and technologies like service orientation, distributed systems, context-aware computing and adaptation of composite systems. Toward achieving this goal, we also present an architecture that underpins the envisioned framework, discuss the current state of our implementation effort, and we outline the open issues and challenges in the field.

1 Introduction

Collective systems comprise heterogeneous entities collaborating towards the achievement of their own objectives, and the overall objective of the collective. Such systems are usually large scale, typically consisting of both physical and virtual entities distributed both organizationally and geographically. In this sense, collective systems exhibit characteristics of both service-oriented and pervasive computing. Furthermore, due to the dynamic nature of the environment they operate in, they have to possess adaptation capabilities.

In our previous work in the ALLOW project, we enabled orchestrations of physical entities [8,16] as the model for individual entities in a collective system. A single entity is modeled using a pervasive flow modeling its functionality, the services it exposes and the functionality a partner entity needs to implement in order to interact with the physical entity. Moreover, the pervasive flows are adaptable in terms of abstract tasks/activities, which can be refined during the execution depending on the goal of the entity. However, this work relies on a model restricting the capabilities of entities to a single behavioral description in terms of Adaptive Pervasive Flows (APFs), and ignores the collaborative aspect in their behavior.

For this purpose, in the current work as part of the ALLOW Ensembles project[1], we aim at defining a Collective Adaptive System (CAS) [19], and the underpinning

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

A.R. Lomuscio et al. (Eds.): ICSOC 2013 Workshops, LNCS 8377, pp. 69–81, 2014.

concepts supporting modeling, execution and adaptation of CAS entities, and their interactions. Toward this goal, we use an approach inspired by biological systems. In particular, we propose to model and manage entities as collections of cells encapsulating their functionality. Entities collaborate with each other to achieve their objectives in the context of ensembles describing the interactions among them.

The contributions of this work can therefore be summarized as follows:

1. Starting from a motivating scenario (Section 2), we introduce a CAS framework (Section 3) defining a conceptual model and the life cycle of systems realizing this model.
2. We introduce an architecture enabling the modeling, execution and adaptation of CAS as distributed, large scale, pervasive systems and we discuss its implementation based on well-established technologies (Section 4).

The paper closes with a summary of related work (Section 5), and concludes with an outline of research challenges and future work (Section 6).

2 Motivating Scenario

Supporting citizens mobility within the urban environment is a priority for municipalities worldwide. Although a network of multi-modal transportation systems (e.g., buses, trains, metro), services (e.g., car sharing, bike sharing, car pooling), and smart technologies (e.g., sensors for parking availability, smart traffic lights, integrated transport pass) are necessary to better manage mobility, they are not sufficient. Citizens must be offered accurate travel information, where and when such information is needed to take decisions that will make their journeys more efficient and enjoyable. In order to deliver "smart services" to citizens, available systems should be interconnected in a synergistic manner constituting a system of systems. The FlexiBus scenario is a case of such system. The goal is to develop a system to support the management and operation of FlexiBuses (FlexiBus Management System (FBMS)), where actors (i.e., passengers, buses, route managers, bus assistance manager etc.) need to cooperate with each other towards fulfilling both individual and collective goals and procedures. As shown in Fig. 1, the system must be able to manage different routes at the same time (e.g. blue and red) set by passengers by allowing pre-booking of pick up points.

More specifically, each *Passenger* can request a trip to one of the predefined destinations in the system, asking to start at a certain time and from a preferred pickup point. The system should manage also special requests from each passenger like traveling with normal or extra sized luggage, or disability related requirements. Each passenger can pay their trip directly in the bus (cash, with a credit card or a monthly pass) or through the FlexiBus company web site. Furthermore, during the route execution, each passenger waiting for a bus can be notified for problems on a selected route (e.g. bus delays, accidents, etc.) Each *Bus Driver* is assigned by the FBMS a precise route to execute, including the list of passengers assigned to it, and a unique final destination (e.g. Trento city center in Fig. 1). During the route realization, each flexibus can also accept passengers that have not booked only if there are available seats. Bus drivers communicate with an assigned *Route Manager* to ask for the next pick-up point and to communicate

Fig. 1. The FlexiBus Scenario

information like passengers check-in. Different routes are created by a *Route Planner* that organizes them to satisfy all passenger requirements (i.e. arrival time and destination) and to optimize bus costs (i.e. shorter distance, less energy consumptions, etc.). To find the set of possible routes, the Route Planner communicates with the *FlexiBus Manager* in order to collect necessary information (i.e. traffic, closed roads, events, etc.) and available resources (i.e. available buses), and to generate alternative routes. A *Bus Assistance Service* is also available for bus drivers to report problems that occur along one route and request for advice/specific activities to be performed (e.g. notify police for an accident, pickup a bus for repair). Finally, a *Payment Service* is the entity that interfaces with various payment systems in order to ensure that ticket purchases are handled correctly.

The system needs to deal with the *dynamic* nature of the scenario, both in terms of the *variability of the actors involved and of their goals*, and of the exogenous *context changes*, e.g. bus damages, passenger requests cancellations, traffic jams, roads closed due to accidents, etc. affecting its operation. Moreover, some of the tasks executed by the actors *require customization* for different environmental situations, like passenger preferences and requirements (e.g. payment with cash or credit card, trip together with a friend, etc.).

3 Overall Framework for CAS

In this section we present our framework to model and execute Collective Adaptive Systems like the FBMS described above.

3.1 Conceptual Model

We model a CAS as a set of *entities* that can collaborate with each other in order to accomplish their business objectives and in some cases common objectives, and for

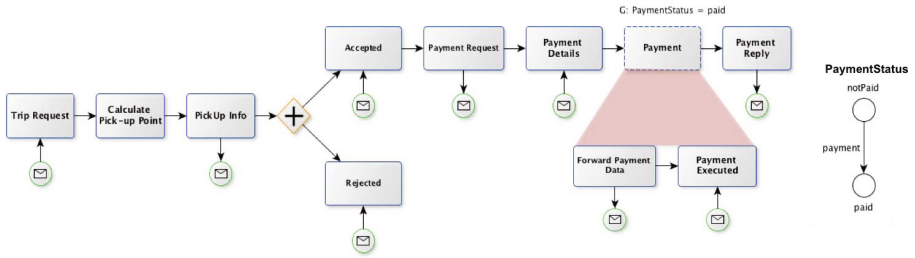


Fig. 2. The Trip Booking Cell Flow

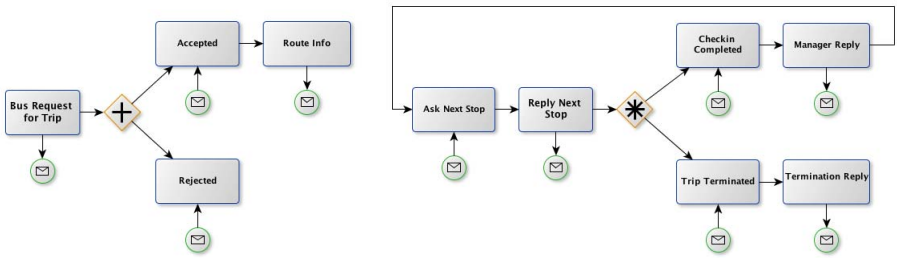
that form one or more *ensembles*. Moreover, to enable interaction among entities, each entity exposes one or more *cells*.

Cells are uniquely identifiable building blocks representing a concrete functionality in a larger, multi-cellular system. Implementing the functionality may involve interacting with other cells through pre-defined protocols. Therefore each cell is defined in terms of its *behavior* (flow) and *protocol*, describing the interaction with other cells and exposed process fragments [15]. For example, the passenger trip booking in the FlexiBus scenario is performed by a specific functionality of the Route Manager entity and it is an example of a cell in the FBMS (see Fig. 2). Among the activities that comprise this flow is Payment, which is marked as an *abstract activity*, in the sense that it requires another cell, or a composition of cells, to implement this functionality. Selecting these cells can be done either during design or run time of the cell at hand.

Cells can be created from each other through differentiation. *Cell differentiation* is the process of modifying/adapting the protocol or flow of an existing cell, resulting in a new cell with more specific functionality. Differentiation can take place either during the instantiation of the cell, or during its lifetime (i.e. in runtime). Accepting only credit cards as part of the Payment activity in Fig. 2 is a case of cell differentiation from the generic cell able to handle different payment options into a cell with more specific functionality. The actual functionality of the Payment activity can actually be provided by another cell, e.g. by the Payment Manager/Service.

After instantiation in the CAS cell instances belong to distinct *entities* and each cell instance belongs to exactly one entity. An entity is a physical or virtual organizational unit aggregating a set of cells. Cells can either be unique in an entity, or they can be replicated by the entity through instantiation as many times as necessary. The Route Manager in the FlexiBus scenario, for example, is an entity containing the Trip Booking cell (Fig. 2) and a Route Assignment cell (Fig. 3a) managing the execution of the route. Each entity has a *context* in which it operates, expressed as a set of stateful properties representing the status of the environment of the entity, e.g. PaymentStatus in Fig. 2. The entity context is accessible and shared by its cells and cells may keep cell specific context. In addition, an entity has a set of *goals*, e.g. ensure that the PaymentStatus context property is set to "paid" at the end of the cell flow execution, that it attempts to fulfill by initiating or participating in one or more ensembles.

An *ensemble* is a set of cells from different entities collaborating with each other to fulfill the objectives of the various entities. Each ensemble is initiated and terminated



(a) Route Assignment Cell Flow (Route Manager)

(b) Route Execution Cell Flow (Bus Driver)

Fig. 3. Examples of Cells and Associated Entities

by one entity, but more than one entities are expected and allowed to join and leave through the ensemble's lifetime. The Route Assignment cell of the Route Manager entity (Fig. 3a) for example, forms an ensemble with the Route Execution Cell of the Bus Driver entity (Fig. 3b) to successfully coordinate the two entities in executing a (FlexiBus) route. Note that one entity may be involved in more than one ensembles simultaneously.

3.2 Lifecycle

The lifecycle of ensembles is depicted in Fig. 4. We distinguish two major phases: design time and run time. During the *design time* phase the ensembles of a CAS are modeled as choreographies and the cells are expressed as Adaptive Pervasive Flows (APFs) [8]. Modeling choreographies implies defining the visible behavior of the participants (i.e. cell protocols), the sequence of exchanged messages, and the types of the exchanged data. During the *Generation & Refinement* step the resulting choreography definition is first transformed into APF skeletons — one for each participant — which also contain the functionality required to support the defined interaction protocol (i.e. sending and receiving messages from partners, data structures for storing the data, etc.). In the subsequent refinement, each APF is edited so that it is completed to an executable APF. Note that the design time phase of choreography subsumes the design time for APFs, i.e. participant implementations/processes. Any kind of adaptation during the design phase of APFs realizes a differentiation of cells. The possible adaptation actions are inserting, deleting and substituting activities and control flow connectors in the APF, changing the data dependencies, editing the context model, and injecting a process fragment that specifies the functionality of an abstract activity.

The *deployment* step uses the APF skeletons from the previous step, their service interfaces, and deployment information about the binding strategies for each of the services to be used. After the deployment the choreography can be executed collectively by APF instances, i.e. the APFs are made available for instantiation by the execution environment. The instantiation of one of the APFs initiates the choreography, which is the beginning of the *run time phase* for the choreography. More than one APF model may be designated as an initiating one, e.g. the Route ensemble may be initiated by a

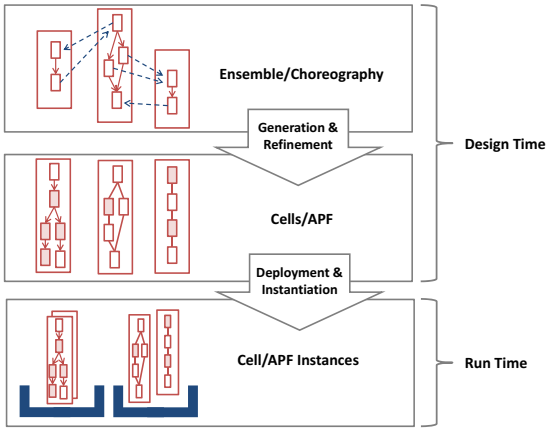


Fig. 4. Lifecycle of Ensembles

cell of the bus or by a cell in a passenger entity. However, if an instance of one APF initiates a choreography, instances of the other participating APFs can only join the initiated choreography, e.g. if a bus cell has started the Route ensemble, passengers can only join the initiated choreography following the predefined rules for passenger check in.

The choreography is completed successfully when the objectives of the entities participating in the ensemble are achieved through executing all APFs in it successfully, or even if some of the cells/APFs have abandoned the ensemble, e.g. if a passenger leaves the bus and moves to another transportation vehicle due to changes in their objectives. For the latter case, fault handling and/or adaptation steps may need to be performed. A choreography is completed abnormally if all participant APFs have been terminated. In this case either the choreography has reached a state for which a termination has been predefined (e.g. the bus breaks down and there is no available one to substitute it, therefore passengers have to join another ensemble, i.e. wait for the next FlexiBus or use an alternative transportation means), or none of the fault handling and/or adaptation steps have been able to complete the choreography successfully. The runtime phase subsumes monitoring and adaptation of choreographies, as well as the runtime and monitoring and adaptation phases of APFs. Adaptation of choreographies is done through adaptation of the visible behavior of the cells and through a change of the interaction protocol among them, including message exchange sequence and message types. Adaptation of an APF may not entail adaptation of the choreography.

4 Realization

4.1 Architecture

The architecture for the modeling and execution of CAS comprises two major component groups (see Fig. 5) which cover the phases of the CAS lifecycle discussed in Section 3.2. More specifically, the *Modeling Tool* comprises three major components:

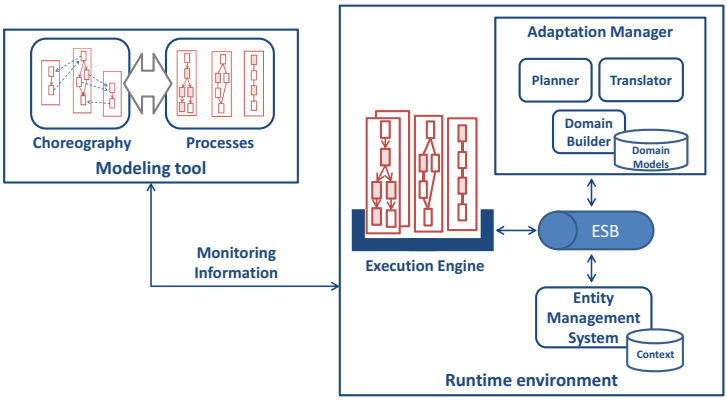


Fig. 5. Architecture overview

a *Choreography Modeler* to create choreography models for the ensembles, a *Transformer* to generate the APF skeletons that can be completed to executable processes by the participant organizations using the APF Editor component, and an *APF Editor* (also called process editor) to allow the visualization and modification of APF models.

The *Runtime Environment* enacts the choreographies. In particular this means that the resulting executable APF models are deployed on one or more *Execution Engines* and can be instantiated at any time. The *Deployment & Instantiation* steps are implementation-specific for each Execution Engine. In order to support the execution of APFs containing abstract activities, the Execution Engine has to be able to start the execution of incomplete processes, allowing the injection of additional activities into APFs. Furthermore, the Execution Engine has to provide fault handling capabilities, both for pre-defined fault and compensation handlers in the APF models, and for failures in the Runtime Environment like service failures and unavailability of other components in the Environment. The Execution Engine has to support user-defined ad hoc control flow changes (e.g. deletion, insertion, substitution of one more activities in the flow). Some of these adaptations require one or more planning steps, for example, in order to resolve abstract activities into concrete ones and to handle the reaction to not pre-modeled faults occurring during the execution of the APF. The component providing this planning functionality is the *Adaptation Manager*.

Once the Adaptation Manager is notified about an execution problem, a change in the context or goals of cells, it decides on the adaptation strategy to be used (horizontal adaptation, vertical adaptation, other adaptation strategies etc. [9]). The choice of the adaptation strategy determines the adaptation goal, which is passed to the *Domain Builder* together with the information about the current context. The *Domain Builder* builds an initial version of the adaptation problem consisting of a context model, a set of available annotated fragments, current context configuration (i.e. the state of context properties), and a set of goal context configurations. The Domain Builder extracts all necessary specification from a repository of *Domain Models*. Taking into account the current context and adaptation goals, the Domain Builder simplifies the context model by pruning all unreachable configurations and removes all services that are useless for

the specified goal. With this optimization the size of the planning domain is significantly reduced. The *Translator* component translates an adaptation problem into a planning problem, which is resolved by the *Planner*. It is also responsible for transforming the results of the Planner into executable APF fragments. Finally, the resulting APF fragment expressing the actions necessary for realizing the adaptation strategy is sent to the Execution Engine, that integrates it into the APF instance.

The *Entity Management System (EMS)* deals with all aspects of entity management: persistence storage and management of APF models and associated entities, access control of APF models and instances, and context provisioning and management. When the EMS creates a new entity, it deploys the entity APFs to the execution engine, adds corresponding context properties to the entity context model, and puts all the entity-related specifications (such as fragments models and the context property diagrams provided by the entity) into the Domain Models storage. When the entity "exits" the CAS, inverse actions are performed. The EMS is responsible for storing the system context (i.e. a set of context properties of all active entities) and constantly synchronizes its current configuration with the application domain by monitoring the environment of the entity. Note that the system context is a simplified view of the application domain. The EMS allows the Adaptation Manager to access the APF models and instances needed for the planning step. Context information is used by the Execution Engine for different purposes: as part of the execution of the APFs, as a trigger for adaptation, and as a configuration parameter for the planning step.

All components (Execution Engine, EMS, Adaptation Manager) should be provided as services and communicate through an *Enterprise Service Bus (ESB)* solution to facilitate their integration. Given the fact that multiple organizational domains may use the *Runtime Environment*, it is necessary to offer multi-tenancy capabilities out of the box for all components in the Environment. Furthermore, the Runtime Environment may contain more than one instances of its components, distributed across on-premises and off-premises Cloud infrastructures, for scalability purposes. This has to be taken into consideration during the integration of the individual components.

4.2 Implementation

In the following we present the status of the implementation of the presented architecture. In particular, we have developed the modeling tool as an Eclipse Graphical Editor. For purposes of expressing choreographies we use the BPEL4Chor language [13] (which is an extension of the WS-BPEL language), and WS-BPEL [25] for implementing the APFs. The user can model the participants in the choreography/ensemble as separate entities and define the interaction among them, including the abstract data types used and the sequence of exchanged messages. BPEL4Chor code is automatically generated by the tool for the choreography, for the list of participants in the choreography and the data exchanged among them. The components implementing the transformation from choreography definition in BPEL4Chor to BPEL process skeletons for each participant and their service interfaces in WSDL, presented in [32], are part of the tool as well as the Eclipse perspective for modeling and editing BPEL processes. The BPEL modeling perspective is an extension of the BPEL Eclipse designer [28]. It is used to view the BPEL skeletons and include additional process elements in order to define the

participants implementation of the choreography role (e.g. bus, passenger, route manager processes). This manual refinement step is simplified by allowing to use predefined process fragments, which are available in the tool catalogue and stored and managed in the process fragment library Fragmento [27].

Additionally, we have extended the tool with a monitoring component for processes, so that during the execution of the APF instances the user can view their status and also adapt manually the instance that is currently being monitored. For this purpose the modeling tool uses run time information from the execution engine provided via its monitoring component. The interaction between the modeling tool, monitoring component and execution engine supports also the runtime adaptation of APFs processes using mechanisms like control flow change (inserting, deleting or substituting process activities and control connectors), changes in the data used in the process instance, and triggering re-execution of some of the already executed activities through [29].

The additional tasks of the Adaptation Manager component are realized by ASTRO-CAptEvo[2] [26], a comprehensive framework for defining highly adaptable service-based systems (SBSs) and supporting their context-aware execution. It can deal with two different adaptation needs: the need to refine an abstract activity within a process instance (i.e. vertical adaptation), and the need to resolve the violation of a context precondition of an activity that has to be executed (i.e. horizontal adaptation). In the second case, the aim of adaptation is to solve the violation by bringing the system to a situation where the process execution can be resumed. Both adaptation mechanisms rely on sophisticated AI planning techniques for the automated composition of services [5]. Moreover, it is able to execute complex *adaptation strategies* that are realized through combining a few adaptation mechanisms and executing them in a precise order, enabling support for addressing complex adaptation problems that cannot be resolved by a single adaptation mechanism [10].

The execution engine for APFs, i.e. the executable processes of the participants in the choreography, is an extended Apache ODE Engine[3], an open source implementation of BPEL. We have extended the engine to support the integration with the modeling tool for the purposes of monitoring, the adaptation mechanisms mentioned above as well as with the ability to stop, suspend and resume a process instance in the engine from the modeling tool [28]. For the ESB component of the architecture we use the ESB^{MT} multi-tenant aware ESB solution, as presented in [30,31]. ESB^{MT} enhances the Apache ServiceMix solution[4] with multi-tenant communication support within service endpoints deployed in the ESB, and multi-tenant aware dynamic endpoint deployment and management capabilities.

The Entity Management System manages all active entities within a CAS. Currently both the entity management and context management parts of the EMS are under construction. Our CAS modeling tool is also missing features supporting modeling of context in the choreographies and APFs. Adaptation mechanisms performing a reaction to context change or driven by context information are also not yet designed and implemented. Our execution engine prototype does not currently support the injection of

[2] `http://www.astroproject.org/captevo`
[3] Apache ODE: `http://ode.apache.org/`
[4] Apache ServiceMix: `http://servicemix.apache.org`

fragments directly into the process instance; note that this is possible for the design time phase. This is due to the fact that the previously presented implementation [26] of this mechanism needs to be integrated in the current implementation. Currently we are also working towards implementation of multi-tenancy of the APF execution engine.

5 Related Work

Collective or adaptive aspects of complex systems have been studied in various domains. For example in *Swarm Intelligence* entities are essentially homogeneous and are able to adapt their behavior considering only local knowledge [11,22]. In existing systems from *Autonomic computing* the entity types are typically limited and the adaptation is guided by predefined policies with the objective to optimize the system rather than evolve it [1,7,23]. In *Service-based systems* utilized on *Internet of Things*, entities are hidden behind the basic abstraction of services, which are designed independently by different service providers, and approaches to automatically compose services to achieve a predefined goal like user specific [18] and/or business goals [24] are the focus. *Multi-agent based systems* concentrate on defining the rules (norms) for regulating the collective work of different agents [12,21]. Most of the results obtained in these domains are tailored to solve problems specific for the domain at hand using a specific language or model but do not present a generic solution for all aspects of collective adaptive systems.

Different *choreography modeling* approaches have been proposed in [3,14,17,20]. Two key approaches followed when modeling choreographies are interaction and interconnection modeling [3]. The former has interaction activities supporting atomic interactions between participants, while the latter interconnects the communication activities of each participant in a choreography. WS-CDL [17] is a choreography language following the interaction modeling approach. It exhibits however a strong dependency between semantic and syntactic aspects, specifically in the definition requirement of message exchange formats between participants at design time [4], lacks support for describing choreographies with an unknown participants number [20], and does not define guidelines for mapping between the choreography modeling language and existing orchestration languages, such as WS-BPEL [25]. The Savara[5] project for example is based on behavior specification and choreography specification using WS-CDL, and behavior simulation, and generation and implementation of business processes using BPEL and Web services. Despite the similarities in some of the used technologies with our approach however, and due to the use of the interaction modeling approach requiring explicit specifications of choreographies and orchestrations, the Savara approach does not allow for dynamically joining and leaving the choreography.

An example of an interconnection modeling approach is the CHOReOS Integrated Development and Runtime Environment which focuses on the implementation and enactment of ultra large scale choreographies of services[6]. By exploiting the notion of models and models@runtime [6] techniques, the CHOReOS Environment provides support

[5] `http://www.jboss.org/savara`

[6] CHOReOS: Large Scale Choreographies for the Future Internet: `http://www.choreos.eu/`

for a top-down and cross-cutting choreographies incorporating the design, enactment, and adaptation of services during runtime. The adaptation requirements addressed in the CHOReOS Environment (react to participants unavailability, or when the SLA is not accomplished) are only a subset of the requirements on ensembles, where context changes in pervasive environments, structural changes in the ensemble, or cells leaving the ensemble, adapting to utility fluctuations etc. are of interest. In the scope of the Open Knowledge European project[7], the interconnection modeling approach is supported by using the Multiagent Protocol (MAP) Web service choreography language for specifying the interaction between peers, which are connected to the services participating in the choreography. Services must be deployed prior to the choreography enactment and the MAP language does not focus on adaptation features. These features present clear deficits with respect to modeling CAS adaptation and the runtime reaction to changes in a service-oriented pervasive environment.

The interaction modeling approach called BPEL^{gold} [20] is based on BPEL4Chor [14]. The coordination logic of participants in choreographies is enabled by an ESB. Both BPEL4Chor and BPEL^{gold} decouple the choreography specification from communication specific details, allowing for dynamic ensemble adaptation during runtime. However, while these approaches possess the required flexibility for defining ensembles no execution environment is currently available for them.

6 Conclusion and Future Work

Collective Adaptive Systems (CAS) are characterized by heterogeneous entities that can join and leave the system at any time towards fulfilling their own objectives. These entities may be physical or virtual, and interact with each other as part of the collective. CAS systems are naturally distributed, both in terms of the participating entities (i.e. geographical location and/or organizational affiliation), and the required infrastructure to support them. In order to enable CAS exhibiting these properties, in this work we introduce a conceptual model inspired by biological systems which comprises collections of cells (functional building blocks) organized into entities (organizational units), interacting with each other in ensembles (collaborations between cells).

In order to discuss the realization of this model, we map its elements to existing technologies and present a lifecycle for the ensembles based on them. We also introduce an architecture for a CAS that ensures complete coverage of the lifecycle, and present the current status of its implementation. Future work focuses on creating an improved context model and provisioning techniques for entities participating in ensembles in different application domains, e.g. in eScience [2], and managing the adaptation of choreographies. Consequently, the components of the prototype implementation discussed in the previous sections have to be extended, and all the remaining components integrated. In addition, different distribution and deployment options for the Runtime Environment will be investigated in order to identify the optimal solution for different CAS.

Acknowledgment. This work is partially funded by the FP7 EU-FET project 600792 ALLOW Ensembles.

[7] Open Knowledge: `http://www.openk.org/`

References

1. Abeywickrama, D.B., Bicocchi, N., Zambonelli, F.: SOTA: Towards a General Model for Self-Adaptive Systems. In: WETICE, pp. 48–53 (2012)
2. Andrikopoulos, V., Gómez Sáez, S., Karastoyanova, D., Weiß, A.: Towards Collaborative, Dynamic & Complex Systems. In: Proceedings of SOCA 2013. IEEE (December 2013) (to appear)
3. Barker, A., Walton, C.D., Robertson, D.: Choreographing Web Services. IEEE Transactions on Services Computing 2, 152–166 (2009)
4. Barros, A., Dumas, M., Oaks, P.: A Critical Overview of the Web Services Choreography Description Language (WS-CDL). BPTrends (March 2005), http://www.bptrends.com/
5. Bertoli, P., Pistore, M., Traverso, P.: Automated composition of Web services via planning in asynchronous domains. Artif. Intell. 174(3-4), 316–361 (2010)
6. Blair, G., Bencomo, N., France, R.B.: Models@run.time. Computer 42, 22–27 (2009)
7. Bruni, R., Corradini, A., Gadducci, F., Lluch Lafuente, A., Vandin, A.: A Conceptual Framework for Adaptation. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 240–254. Springer, Heidelberg (2012)
8. Bucchiarone, A., Lafuente, A.L., Marconi, A., Pistore, M.: A Formalisation of Adaptable Pervasive Flows. In: Laneve, C., Su, J. (eds.) WS-FM 2009. LNCS, vol. 6194, pp. 61–75. Springer, Heidelberg (2010)
9. Bucchiarone, A., Marconi, A., Pistore, M., Raik, H.: Dynamic Adaptation of Fragment-Based and Context-Aware Business Processes. In: Proceedings of ICWS 2012, pp. 33–41 (2012)
10. Bucchiarone, A., Marconi, A., Pistore, M., Traveso, P., Bertoli, P., Kazhamiakin, R.: Domain Objects for Continuous Context-Aware Adaptation of Service-based Systems. In: Proceedings of ICWS 2013, pp. 571–578 (2013) (to appear)
11. Pinciroli, C., et al.: ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. In: Proceedings of IROS. pp. 5027–5034 (2011)
12. Cabri, G., Puviani, M., Zambonelli, F.: Towards a taxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In: CTS, pp. 508–515 (2011)
13. Decker, G., Kopp, O., Leymann, F., Pfitzner, K., Weske, M.: Modeling Service Choreographies Using BPMN and BPEL4Chor. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 79–93. Springer, Heidelberg (2008)
14. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for Modeling Choreographies. In: Proceedings of ICWS 2007 (2007)
15. Eberle, H., Unger, T., Leymann, F.: Process Fragments. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2009, Part I. LNCS, vol. 5870, pp. 398–405. Springer, Heidelberg (2009)
16. Herrmann, K., Rothermel, K., Kortuem, G., Dulay, N.: Adaptable Pervasive Flows - An Emerging Technology for Pervasive Adaptation. In: Proceedings of PerAda 2008. IEEE (2008)
17. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., Barreto, C.: Web Services Choreography Description Language Version 1.0 (November 2005)
18. Kazhamiakin, R., Paolucci, M., Pistore, M., Raik, H.: Modelling and Automated Composition of User-Centric Services. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010, Part I. LNCS, vol. 6426, pp. 291–308. Springer, Heidelberg (2010)
19. Kernbach, S., Schmickl, T., Timmis, J.: Collective Adaptive Systems: Challenges Beyond Evolvability. ACM Computing Research Repository (CoRR) (August 2011)

20. Kopp, O., Engler, L., van Lessen, T., Leymann, F., Nitzsche, J.: Interaction Choreography Models in BPEL: Choreographies on the Enterprise Service Bus. In: Fleischmann, A., Schmidt, W., Singer, R., Seese, D. (eds.) S-BPM ONE 2010. CCIS, vol. 138, pp. 36–53. Springer, Heidelberg (2011)
21. Lavinal, E., Desprats, T., Raynaud, Y.: A generic multi-agent conceptual framework towards self-management. In: NOMS, pp. 394–403 (2006)
22. Levi, P., Kernbach, S.: Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution. Springer (2010)
23. Lewis, P., Platzner, M., Yao, X.: An outlook for self-awareness in computing systems. Awareness Magazine (2012)
24. Marconi, A., Pistore, M., Traverso, P.: Automated Composition of Web Services: The ASTRO Approach. IEEE Data Eng. Bull. 31(3), 23–26 (2008)
25. OASIS: Web Services Business Process Execution Language Version 2.0 (April 2007)
26. Raik, H., Bucchiarone, A., Khurshid, N., Marconi, A., Pistore, M.: ASTRO-CAptEvo: Dynamic Context-Aware Adaptation for Service-Based Systems. In: Proceedings of SERVICES, pp. 385–392 (2012)
27. Schumm, D., Karastoyanova, D., Leymann, F., Strauch, S.: Fragmento: Advanced Process Fragment Library. In: Proceedings of ISD 2010, pp. 659–670. Springer (2010)
28. Sonntag, M., Hahn, M., Karastoyanova, D.: Mayflower - Explorative Modeling of Scientific Workflows with BPEL. In: Proceedings of the Demo Track of BPM 2012. CEUR Workshop Proceedings, pp. 1–5 (2012)
29. Sonntag, M., Karastoyanova, D.: Ad hoc Iteration and Re-execution of Activities in Workflows. International Journal on Advances in Software 5(1&2), 91–109 (2012)
30. Strauch, S., Andrikopoulos, V., Leymann, F., Muhler, D.: ESB^{MT}: Enabling Multi-Tenancy in Enterprise Service Buses. In: Proceedings of CloudCom 2012, pp. 456–463. IEEE Computer Society Press (December 2012)
31. Strauch, S., Andrikopoulos, V., Sáez, S.G., Leymann, F., Muhler, D.: Enabling Tenant-Aware Administration and Management for JBI Environments. In: Proceedings of SOCA 2012, pp. 206–213. IEEE Computer Society Conference Publishing Services (December 2012)
32. Weiß, A., Andrikopoulos, V., Gómez Sáez, S., Karastoyanova, D., Vukojevic-Haupt, K.: Modeling Choreographies using the BPEL4Chor Designer: An Evaluation Based on Case Studies. Tech. Rep. 2013/03, IAAS, University of Stuttgart (2013)