Context-Aware Programming for Hybrid and Diversity-Aware Collective Adaptive Systems

Hong-Linh Truong(✉) and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology, Vienna, Austria
{truong,dustdar}@dsg.tuwien.ac.at

Abstract. Collective adaptive systems (CASs) have been researched intensively since many years. However, the recent emerging developments and advanced models in service-oriented computing, cloud computing and human computation have fostered several new forms of CASs. Among them, Hybrid and Diversity-aware CASs (HDA-CASs) characterize new types of CASs in which a collective is composed of hybrid machines and humans that collaborate together with different complementary roles. This emerging HDA-CAS poses several research challenges in terms of programming, management and provisioning. In this paper, we investigate the main issues in programming HDA-CASs. First, we analyze context characterizing HDA-CASs. Second, we propose to use the concept of hybrid compute units to implement HDA-CASs that can be elastic. We call this type of HDA-CASs h^2**CAS** (Hybrid Compute Unit-based HDA-CAS). We then discuss a meta-view of h^2**CAS** that describes a h^2**CAS** program. We analyze and present program features for h^2**CAS** in four main different contexts.

1 Introduction

Collective adaptive systems (CASs) have been researched intensively since many years [1–4]. For solving complex problems in business and society, new concepts of CASs have been emerging by utilizing human-based computing and software-based computing elements as basic building blocks of CASs. Among them, the Hybrid and Diversity-aware CAS (HDA-CAS) has emerged as a new type of CAS that consists of diverse machine- and human-based computing elements [5]. HDA-CASs promise a new way to solve complex problems that requires both human knowledge and machine capabilities, such as in simulation, urban planning, and city management.

While CASs can be built based on computing elements from different environments, in our research, we are particularly interested in HDA-CASs that are built from software-based services, thing-based services, and human-based services, following service-oriented and cloud computing models. In such models, machine-based and human-based elements provide fundamental computation, data, and network functions and these elements have well-defined service interfaces and are provisioned under cloud models. Atop diverse types of services

F. Fournier and J. Mendling (Eds.): BPM 2014 Workshops, LNBIP 202, pp. 145–157, 2015.
DOI: 10.1007/978-3-319-15895-2_13

in the cloud, new types of dynamic elasticity properties have emerged. First, a huge amount of diverse types of resources are available that can be taken into the construction of CASs on demand. Second, elasticity requirements from the consumer for whom a CAS is provided and problems being solved by the CAS force us to design CASs capable of handling cost and quality changes. Therefore, we believe that utilizing dynamic, on-demand service units from clouds to establish HDA-CASs is a promising direction.

Since HDA-CASs are designed to deal with complex problems in an adaptive way, examining elasticity mechanisms for building HDA-CASs using cloud resources, offering cloud computing models for HDA-CASs, and dynamically managing HDA-CASs at runtime is a very interesting but challenging problem. In our previous work, we have described cloud models for software and people, basic hybrid compute unit models for establishing computing systems including both software and people, and basic programming APIs for these units [6,7]. In this paper, we examine how such fundamental building blocks can be used to build HDA-CASs. We advocate the form of HDA-CASs being constructed from hybrid compute units (HCUs), which consist of software-, thing- and human-based service units. These units are provisioned under the service concept (with well-defined interfaces and utilization model), enabling dynamic programming features for utilizing them. To this end, this paper presents the following contributions:

- analysis and definition of context associated with HDA-CASs
- analysis of hybridity and elasticity properties of HDA-CASs,
- analysis of the utilization of hybrid compute units for implementing HDA-CASs, called h^2**CAS**,
- a meta-view of main building blocks for h^2**CAS**, and
- analysis of programming features for h^2**CAS** in four main high-level contexts.

Our contributions provide fundamental work for the development of program specification of h^2**CAS**, h^2**CAS** provisioning services, and h^2**CAS** elasticity techniques.

The rest of this paper is structured as follows: Sect. 2 defines the context of HDA-CAS and discusses the hybridity and elasticity of HDA-CASs. Section 3 defines HDA-CASs using hybrid compute units and a meta-view of our h^2**CAS** specification. Section 4 discusses issues in programming h^2**CAS**. Related work is presented in Sect. 5. We conclude the paper and outline our future work in Sect. 6.

2 Analyzing Contexts of HDA-CAS

2.1 Context of HDA-CAS

Our goals to support context-aware programming HDA-CASs are to provide (i) right constructs for specifying what constitutes a HDA-CAS, (ii) tools and middleware for deploying, provisioning and instantiating HDA-CASs based on their specifications, and (iii) means for programming the control and reconfiguration of HDA-CASs at runtime. Therefore, it is important to understand the context in which a HDA-CAS will be formed and operated as well as possible contexts inherent in the lifetime of HDA-CASs. We define a context of a HDA-CAS as follows:

Definition 1 (Context of HDA-CAS). *Context of a HDA-CAS describes situational information about tasks and quality of results (*What*), structures of the HDA-CAS and its constituting units for computation/data/network functions as well as for monitoring/control/management functions (*Who*), and the coordination and elasticity mechanisms that control the operation of the HDA-CAS (*How*) in a determined time frame (*When*).*

To support context-aware programming of HDA-CASs, we need to address several open questions of *When, What, Who/Which* and *How* characterizing contexts in which programming features for HDA-CASs play a crucial role:

- *When:* When a HDA-CAS is formed and instantiated? From when to when a HDA-CAS is in a particular context? When does a HDA-CAS switch its context?
- *What:* What are the tasks that a HDA-CAS has to solve in a particular context? What are the expected quality of results (QoRs) for these tasks?
- *Which/Who:* Which types of units are needed for performing computation/data/network functions and for monitoring/management/control functions in a HDA-CAS context? Which structures can be used to describe a HDA-CAS and its units?
- *How:* How does a HDA-CAS work? How does a HDA-CAS coordinate its units? How does a HDA-CAS support and control its elasticity?

Programming a HDA-CAS means that we need to be able to describe several types of information related to the above-mentioned questions in well-defined specifications. Based on that, via software-defined APIs, we can create, provision

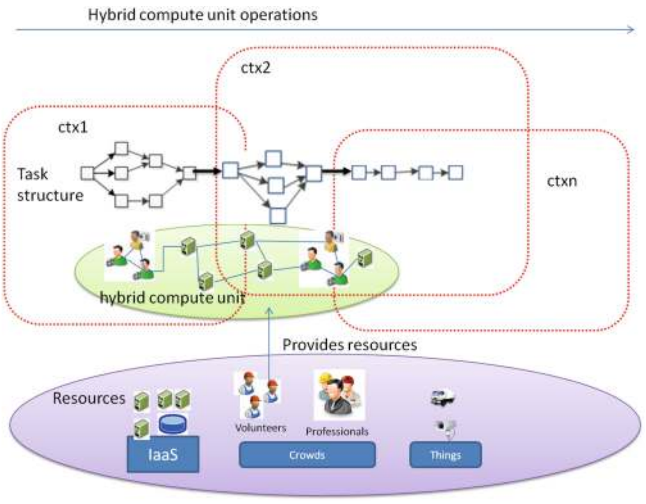

Fig. 1. High-level view of context (`ctx`), task structure and units associated with HDA-CAS

and control HDA-CASs to enable units within these HDA-CASs to interact and perform their tasks/roles.

Let us consider the case in which a HDA-CAS is built by using a hybrid compute unit (HCU) [6,7]. There are complex relationships among the *What* and *Who/Which* within a context and among contexts within the lifetime of a HDA-CAS, as shown in Fig. 1. A HCU is provisioned from different resources (via virtualization techniques). The HCU is the Who/Which of the HDA-CAS. The HCU is intended for solving problems which have complex, and possibly evolving, task structures, dependent on the context. Therefore, under a specific context in the lifetime of the HDA-CAS, HCU structures may be changed to assure the QoR associated with the tasks.

2.2 Hybridity in CASs

Hybridity is an intrinsic property of HDA-CASs. This property is due to the fact that HDA-CAS is formed to address complex problems, which require us to employ diverse and hybrid types of units and roles:

- Different types of resources, including machine-based, human-based and thing-based resources that offer computation, data, and network functions as well as monitoring, control and management functions.
- Different roles performed in the same collective: including performing computation/data/network functions and supporting management, monitoring and control functions. Different types of resources or a single resource might perform different roles or the same role at different time based on different capabilities.

Therefore, techniques for programming HDA-CASs must support fundamental programming constructs and algorithms to deal with the hybridity of computing models in HDA-CASs. In terms of hybridity, the following aspects are important:

- We must be able to execute, coordinate, and manage computation/data/network functions using hybrid processing units (e.g., CPU/core for machine-based computing, human brain for human-based computing, and sensor for thing-based computing),
- We must be able to program hybrid architectures (e.g., the cluster of machines for machine-based computing, the individual/team for human-based computing, and the web of things for thing-based computing), and
- We must be able to program hybrid communication protocols (e.g., TCP/IP for machine-based computing, social network for human-based computing, and MQTT for thing-based computing).

A HDA-CAS will consists of a mixture of these processing units, architectures and communications.

2.3 Elasticity in HDA-CASs

One of the main issues in programming and provisioning HDA-CAS is to support the elasticity principles, covering resource, quality, and cost/benefit elasticity. These principles should be the core mechanisms for, e.g., managing and controlling the operations of HDA-CASs (thus, addressing the *How* aspect of HDA-CAS contexts). The main reason is that HDA-CAS is a dynamic entity whose structures, tasks, and QoRs are dynamically changed. The type of resource elasticity can be seen through the change, reduction and expansion of units for computation/data/network functions. Other types of elasticity can be observed through the following aspects:

- Mixture of different QoRs from a single collective, given a specific goal: a collective is dynamic w.r.t. structures, interactions, and performance, thus, it can produce different QoRs, depending on different settings (e.g., time, availability, incentives, to name just a few).
- Mixture of cost/benefit models: a collective might perform a goal (offering a capability) with different cost/benefit models with different or the same quality.

Such elasticity capabilities must be captured, modeled and associated with HDA-CASs, enabling the management and control of HDA-CASs via programming features. For example, we must be able to program the selection and utilization of suitable units for different types of tasks and QoRs (e.g., specifying expected performance, cost, and quality of data). Overall, we foresee the following levels of elasticity:

- *Processing Units:* The basic mechanisms are (i) to add/remove new processing units based on the load and (ii) to replace existing processing units with new processing units. If units are humans, then we can search clouds of human-based services to find relevant units. If it is software then we can find new software based on service selection techniques.
- *Architecture:* the architecture reflects how different units performing computation/data/network functions and monitoring/control/management functions can be glued. The basic mechanisms are (i) to provision different static and runtime topologies for different types of units, and (ii) to change different protocols/algorithms within monitoring/control/management units.
- *Communications:* There will be multiple communication protocols among different types of units, e.g., communications among units performing computation/data/network functions and among monitoring/control/management units. The basic mechanisms are (i) adding/removing communication protocols, (ii) reconfiguring existing protocols, and (iii) replacing existing protocols with new protocols.

3 h^2CAS– HDA-CAS Using Hybrid Compute Units

3.1 Hybrid Compute Units and HDA-CAS

Based on the two main properties of hybridity and elasticity of HDA-CASs, to specify and program HDA-CAS's structures, we rely on the hybrid compute unit (HCU) concept – a unified model that is able to capture different types of service units and their relationships. Using service units and relationships modeled in the HCU, the HDA-CAS programmer can define HDA-CAS structures, including topologies and communications, and configure other elements, such as algorithms for selecting units for performing computation/data/network functions, for evaluating quality of results, and for controlling the elasticity of units by considering costs and benefits. The HCU model is described in detail in [7]. Service units are associated with elasticity capabilities; each capability can be programmed via software-defined APIs. A general concept is that the consumer of HDA-CASs acquires a HCU representing the HDA-CAS structure. Then consumer can control the elasticity of the HDA-CAS via APIs, triggering suitable set of actions, each mapped to some primitives of units. We define a model of HDA-CAS based on the HCU as follows:

Definition 2 (HCU-based HDA-CAS). *A HCU-based HDA-CAS (h^2**CAS**) includes a set of service units which can be software-based services, human-based services and thing-based services that can be provisioned, deployed and utilized as a collective on-demand based on different quality, pricing and incentive models.*

In our work, programming h^2**CAS** means that: (i) we are able to specify relevant information of h^2**CAS**, (ii) h^2**CAS** specification will be compiled and executed by some middleware, and (iii) h^2**CAS** operations will be controlled at runtime by the h^2**CAS** itself or by external controllers via software-defined APIs. Main programming features that a h^2**CAS** programming framework should support:

- *Initialization:* we must be able to describe the structure of h^2**CAS** including units, architectures and communications. The *Who/Which* must be structured based on types of the tasks and the QoRs (the *What*). We must be able to initiate the h^2**CAS** based on the structure, deployment, and configuration.
- *Elasticity Management:* we must be able to understand elasticity contexts of h^2**CAS**, which must be monitored. The elasticity of h^2**CAS** will be used to control dynamic changes of h^2**CAS** structures to meet the expected QoR. During runtime, depending on specific contexts, we can measure/monitor/predict QoRs and perform elasticity actions by calling elasticity APIs of h^2**CAS**.

3.2 Meta-Program for h^2CAS

To specify h^2**CAS**, we need to determine the types of units needed for performing computation/data/network functions and for performing management/control/monitoring functions, possible coordination and communication protocols/models among different types of units and elasticity capabilities of these units to control the elasticity of h^2**CAS**. They can be determined before the initialization of a h^2**CAS** or during the lifetime of a h^2**CAS**, based on specific contexts of h^2**CAS**. In this section, we discuss h^2**CAS** meta-view and leave the detailed design and implementation of the meta-view specification for the future work.

Figure 2 presents a meta-view of h^2**CAS** programs describing possible service units and interactions. h^2**CAS** includes four main building block specifications:

- *Task Management:* three main types of units should be specified for task management. InputTaskStorageUnit is used to manage input tasks (e.g., from the consumer of h^2**CAS**) that need to be solved by h^2**CAS**. TaskMatchingUnit and TaskControlUnit are used to match tasks to units performing computation/data/network functions and to manage tasks performed by such units, respectively.
- *Result Management:* at least three main types of units should be specified. OutputResultStorageUnit is used to manage results of h^2**CAS** that will be sent to the consumer. QoREvaluationMatchingUnit and QoREvaluation ControlUnit are used to find units for performing the QoR evaluation and manage QoR evaluation tasks.
- *Computation/Data/Network Task Execution and Control:* types of possible units used for performing computation/data/networks should be specified. Three main units for managing computation/data/network units (CDNElas ticityControlUnit), for supporting communications (CDNCommunication Unit and for interacting with clouds of services (CloudConnectorUnit) should be specified.
- *Elasticity Monitoring and Control:* two main units should be specified. First, MonitoringUnit is used for performing different monitoring activities, including monitoring task management, result management, computation/data/

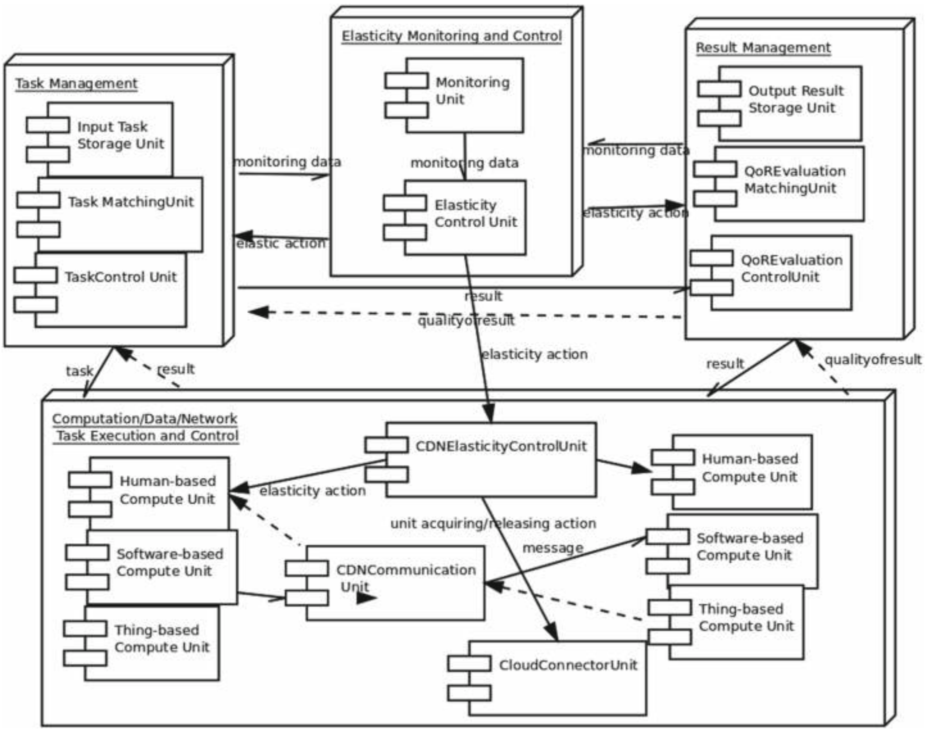

Fig. 2. h^2**CAS** meta-view

network functions, etc. Second, `ElasticityControlUnit` is specified for performing elasticity controls, such controlling units for performing computation/data/network functions, communication units, task management units, etc.

In terms of programming, these types of units are "class" and when initialized, we can have different instances of these units based on different implementations and configurations. Each building block will require different ways to specify and program units, architectures and protocols. Among different building blocks, there will be protocols for their interactions. Similar to the types of units, these protocols can also have different instances. It is important to note that these units can be software-based, thing-based, or human-based. For example, `TaskControlUnit` or `CDNElasticityControl` can be human-based services. Therefore, h^2**CAS** operations are not fully automatically managed and controlled. Instead, these operations are carried out through a combination of human and machine activities.

4 Context-Aware Programming Features for h^2CAS

In the following, we discuss programming features for h^2**CAS** that are centered around four main specific contexts:

- *consumer-generated independent continuous task context:* in this context, a h^2**CAS** is established and used for solving independent tasks sent continuously by the consumer of the h^2**CAS**.
- *consumer-generated dependent task context:* in this context, a h^2**CAS** is established and used for solving a task graph sent by the consumer of the h^2**CAS**.
- *evolving independent task context:* in this context, a h^2**CAS** is established and used for solving a set of independent tasks but during the problem solving time, several new independent tasks are created by the h^2**CAS**.
- *evolving dependent task context:* in this context, a h^2**CAS** is established and used for solving a set of dependent tasks and during the problem solving several dependent tasks are created newly by the h^2**CAS**.

They are high-level contexts which can be subdivided into different types of sub-contexts based on the *What* aspect (e.g., a sub-context in which only the same type of tasks is solved), the *Who/Which* aspect (e.g., a context in which all tasks are solved by human-based compute units), and/or the *How* aspect (e.g., a context in which only cost elasticity is needed).

4.1 Consumer-Generated Independent Continuous Task Context

Context Description. In this context, a h^2**CAS** is provisioned for solving a flow of independent tasks from its consumers. All the tasks are atomic and the tasks are created by the consumer of the h^2**CAS**. QoR is associated with individual tasks. It is possible to have different types of tasks, which require different types of units performing computation/data/network functions. Tasks are continuously given to h^2**CAS**. This kind of task delivery is highly related to works in crowdsourcing tasks [8]. However, the main difference is that the task flow is continuous.

Initialization. For independent tasks, h^2**CAS** will receive a flow of independent tasks which are put into `InputTaskStorageUnit` by the consumer of the h^2**CAS**. We can safely assume that there is only one service unit working on a specific task at a given time. If the QoR of a task does not match the requirement (e.g., the task cannot be finished by a unit), the task should be reassigned and performed by another unit. For this reason, we can also decide not to include `CDNCommunicationUnit` for units performing computation/data/network functions. Thus, h^2**CAS** could be programmed by forming h^2**CAS** based on general task descriptions without `CDNCommunicationUnit`.

A minimum set of units for performing tasks can be established by programming/configuring suitable (pre-)runtime/static unit formation algorithms within `CDNElasticityControlUnit`. The h^2**CAS** can coordinate task execution based on different coordination models programmed in `TaskControlUnit`.

Elasticity. Depending on the task types, h^2**CAS** could be deployed to use only clouds of human-based services or clouds of hybrid services. At runtime, based on monitoring information, especially QoR (e.g., higher or lower QoR than expected), `ElasticityControl` and `CDNElasticityControlUnit` can apply elasticity controls to individual units performing computation/data/network functions, to change coordination protocols within task management and output management units, or to interact with other clouds to negotiate pricing models and acquire/release resources. Elasticity controls can be performed via different control algorithms and can be also programmed via elasticity APIs.

4.2 Consumer-Generated Dependent Task Context

Context Description. This context is quite similar to the previous context in Sect. 4.1, except a complete task graph is given to the h^2**CAS**. The task graph includes dependent tasks, e.g., in terms of data or control dependencies.

Initialization. Similarly to the previous context, a h^2**CAS** can be established with a minimum cost and a limited number of units performing computation/data/network functions. Thus, initially the h^2**CAS** can be just enough for solving a sub-graph and then eventually be extended to solve other tasks, e.g. followed the strategies in [6]. Although tasks are dependent, communications among h^2**CAS** units performing tasks may or may not be established. When `CDNCommunicationUnit` is not needed, the dependencies among tasks can be managed by `TaskManagementUnit`. All task results have to be routed back to the *TaskControlUnit*. Otherwise, the dependencies can also be managed by units performing tasks and using `CDNCommunicationUnit` to send/receive messages about task results to other units. Another benefit of using `CDNCommunicationUnit` is to facilitate the discussion among units performing tasks, when these units are human-based.

Elasticity. The elasticity in this context is similar to the previous context in Sect. 4.1. However, the elasticity of units performing computation/data/network functions could be relied on different strategies, such as expanding and reducing units by considering Business-as-Usual and corrective action cases [6].

4.3 Evolving Independent Task Context

Context Description. In this context, a set of independent tasks needs to be solved by a h^2**CAS**; each task has its own expected QoR. However, solving an independent task might lead to the creation of sub-tasks. This requires two ways of interactions among a unit performing a task in h^2**CAS**, `TaskControlUnit` and `QoREvaluationControlUnit` to decide how sub-tasks should be assigned, executed and evaluated.

Initialization. We could start with a common strategy of initializing a h^2**CAS** with a minimum capability (e.g., based on costs) and later on we can use elasticity mechanisms to expand or reduce the h^2**CAS**. Since a unit performing a task needs to interact with `TaskControlUnit` to decide how to assign, execute and evaluate sub-tasks, two different possibilities can be configured: (i) the unit performing a task returns the outcome – either sub-tasks or the result of the task – to management units in `TaskManagement` and let them to manage the outcome and (ii) the unit performing a task coordinates the execution of sub-tasks it creates. In the latter, the initialization requires `CDNCommunicationUnit` and another protocol to allow the unit to call `TaskMatchingUnit` and `CDNElasticityUnit` to assign and manage its sub-tasks.

Elasticity. When a unit performing computation/data/network functions or QoR evaluation is not responsible for its newly-created tasks, the typical elasticity mechanisms in previous contexts (Sects. 4.1 and 4.2) could be utilized. Otherwise, different elasticity mechanisms can be performed by the unit by calling `CDNElasticityControlUnit`. When the unit is a human-based compute unit, the elasticity actions are manually done by the human. In case, the unit is a software-based compute unit, we need to program suitable algorithms for executing, managing and evaluating sub-tasks. Note that when a unit utilizes elasticity controls to acquire other units to perform its sub-tasks, it is possible that the newly acquired units are the core elements for a new h^2**CAS**. In other words, a unit can create a new h^2**CAS** to solve its sub-tasks.

4.4 Evolving Dependent Task Context

Context Description. In this context, a set of dependent tasks is given to h^2**CAS**. Usually the number of tasks is small but the complexity of the task is high. Furthermore, QoR is associated with the whole set of tasks. While solving tasks, new tasks, also dependent on other tasks, could be created. Overall, the task graph will be expanded and reduced until the h^2**CAS** completes all the tasks in the graph.

Initialization. A h^2**CAS** could be formed based on the task graph using strategies similar to thos in Sect. 4.3.

Elasticity. The elasticity can be carried out in a similar way to the context in Sect. 4.3. Since the QoR is associated with the whole task and tasks are strongly dependent by each other, elasticity control mechanisms need to be programmed in such a way that takes into account these strong dependencies.

5 Related Work

In [1], a formal model for socio-technical CAS is discussed. It discusses how to specify CAS using different formal models. Generally, there is no software framework for programming h^2**CAS** as we describe in this paper. In the state-of-the art, typically a specific CAS is built with several components and it is used for different purposes by varying inputs into the CAS. Another way to solve complex problems by using human-based services and software-based services is to design a specific middleware/platform to manage and distribute tasks to different resources, which can be software or humans. An example of such platforms is Jabberwocky [9] which allows specifying types of people based on personal properties and expertise and route tasks and combing humans with machines. These platforms are not systems for HDA-CAS. Our work actually aims at generalizing these platforms by providing techniques for programming and provisioning such specific CASs or middleware using service units.

Software architectures describing the interactions between humans and other software service units have been discussed, e.g. in [10]. They aim at supporting design techno-social software systems that allow people to work with machines. Our work is different as, in the programming perspective, we support the developer to write h^2**CAS** program of which, in addition to other types of information, some architectures of software-based and human-based units are specified by the developer.

Several task management models, coordination and communication protocols for collaborative complex problem solving have been developed. For example, two approaches in designing task processes for humans are studied in [8] but managing task processes is just one feature that influences the design and operation of a collective in our work. In fact a major related work in human computation focus on designing task processes and distributing tasks to different human compute units [11,12]. Our work differs from them as we focus on programming systems that enable the execution of different task processes and on the elasticity of these systems to deal with elasticity requirements of tasks. In [13] several researchers have discussed several issues for supporting collaborative, dynamic and complex tasks performed by crowds. In general, they analyze several research challenges and we believe that certain types of collectives built atop human units should address these challenges. Our h^2**CAS** model built for elasticity and based on hybrid compute units could be used to program platforms to support some of these mentioned challenges.

In our previous work, we focus on HCU and service unit models [6,7]. We have presented the hybridity and elasticity of different types of units in HCU in general. In this paper, we examine how to implement h^2**CAS** using HCUs, therefore, our hybridity and diversity analysis of h^2**CAS** is bound to the context from/in which h^2**CAS** is formed and operates.

6 Conclusions and Future Work

HDA-CAS is a new form of collective adaptive systems (CASs) built for solving complex problems by utilizing diverse and hybrid service units offering well-defined cloud provisioning models. However, we need to support the right programming features to simplify the creation, provisioning and execution of HDA-CAS. In this paper we analyze possible contexts associated with HDA-CAS and propose the utilization of hybrid compute units to program HDA-CAS. In doing so, we focused on context aspects – such as *When, What, Who/Which* and *How* – in programming HDA-CASs. We presented h^2**CAS** as one way of implementing HDA-CASs as well as outlined a meta-view for specifying h^2**CAS** and programming features for h^2**CAS** in some general contexts.

Currently, we are working on a specification of programming constructs and models that can be used to specify h^2**CAS** in detail. Furthermore, we are working on tools and middleware for compiling h^2**CAS** specification and deploying, controlling and provisioning techniques for h^2**CAS**.

Acknowledgments. We thank Muhammad Z. C. Candra, Mirela Riveni, Ognjen Scekic and Vincenzo (Enzo) Maltese for fruitful discussions on hybrid compute units, elasticity, and collective adaptive systems. The work mentioned in this paper is partially supported by the EU FP7 SmartSociety project under grant N° 600854.

References

1. Coronato, A., Florio, V.D., Bakhouya, M., Serugendo, G.D.M.: Formal modeling of socio-technical collective adaptive systems. In: Proceedings of the 2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems Workshops. SASOW 2012, pp. 187–192. IEEE Computer Society, Washington, DC, USA (2012)
2. Fundamentals of collective adaptive systems. http://focas.eu/
3. Andrikopoulos, V., Saez, S.G., Karastoyanova, D., Weiss, A.: Towards collaborative, dynamic and complex systems (short paper). In: SOCA, pp. 241–245. IEEE (2013)
4. Bruni, R., Corradini, A., Gadducci, F., Lafuente, A.L., Vandin, A.: Modelling and analyzing adaptive self-assembly strategies with Maude. Sci. Comput. Program. **99**, 75–94 (2015)

5. Hybrid and diversity-aware collective adaptive systems. http://www.smart-society-project.eu/
6. Truong, H.L., Dustdar, S., Bhattacharya, K.: Conceptualizing and programming hybrid services in the cloud. Int. J. Coop. Info. Syst. **22**, 1341003 (2013)
7. Truong, H.-L., Dam, H.K., Ghose, A., Dustdar, S.: Augmenting complex problem solving with hybrid compute units. In: Lomuscio, A.R., Nepal, S., Patrizi, F., Benatallah, B., Brandić, I. (eds.) ICSOC 2013. LNCS, vol. 8377, pp. 95–110. Springer, Heidelberg (2014)
8. Little, G., Chilton, L.B., Goldman, M., Miller, R.C.: Exploring iterative and parallel human computation processes. In: Proceedings of the ACM SIGKDD Workshop on Human Computation. HCOMP 2010, pp. 68–76. ACM, New York, USA (2010)
9. Ahmad, S., Battle, A., Malkani, Z., Kamvar, S.: The jabberwocky programming environment for structured social computing. In: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology. UIST 2011, pp. 53–64. ACM, New York, USA (2011)
10. Dorn, C., Taylor, R.N.: Coupling software architecture and human architecture for collaboration-aware system adaptation. In: Notkin, D., Cheng, B.H.C., Pohl, K. (eds.) ICSE, pp. 53–62. IEEE / ACM, San Francisco (2013)
11. Quinn, A.J., Bederson, B.B.: Human computation: a survey and taxonomy of a growing field. In: Tan, D.S., Amershi, S., Begole, B., Kellogg, W.A., Tungare, M. (eds.) CHI, pp. 1403–1412. ACM, New York (2011)
12. Kulkarni, A.P., Can, M., Hartmann, B.: Turkomatic: automatic recursive task and workflow design for mechanical turk. In: Proceedings of the 2011 Annual Conference Extended Abstracts on Human Factors in Computing Systems. CHI EA 2011, pp. 2053–2058. ACM, New York, USA (2011)
13. Kittur, A., Nickerson, J.V., Bernstein, M., Gerber, E., Shaw, A., Zimmerman, J., Lease, M., Horton, J.: The future of crowd work. In: Proceedings of the 2013 Conference on Computer Supported Cooperative Work. CSCW 2013, pp. 1301–1318. ACM, New York, USA (2013)