Virtualizing Communication for Hybrid and Diversity-Aware Collective Adaptive Systems

Philipp Zeppezauer, Ognjen Scekic, Hong-Linh Truong, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology, Austria
{zeppezauer,oscekic,truong,dustdar}@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at

Abstract. Hybrid and Diversity-Aware Collective Adaptive Systems (HDA-CAS) form a broad class of highly distributed systems comprising a number of heterogeneous human-based and machine-based computing (service) units. These units collaborate in ad-hoc formed, dynamically-adaptive collectives. Whenever possible, the collectives are allowed to self-orchestrate. The flexibility of these collectives makes them suitable for processing elaborate tasks, but at the same time, building a system to manage such collectives is challenging. In this paper, we address the fundamental communication challenges for HDA-CAS. We present the design of a middleware for virtualizing communication within and among collectives. The middleware is able to handle numerous, intermittently-available, human and software-based service units, and manage the notion of collectivity transparently to the programmer. A prototype implementation for validation purposed is also provided.

1 Introduction

Collective Adaptive System (**CAS**) [2] is an umbrella-term denoting distributed systems comprised of multiple autonomous computing elements with different individual properties, but with the fundamental property of collectiveness. *Collectiveness* implies that the individual elements need to communicate and collaborate in order to reach common decisions, or perform tasks jointly. *Adaptiveness* is another basic property of CASs, implying open systems where computing elements may join and leave, and dynamically alter collective compositions or task execution goals. CASs come in a variety of forms. **Hybrid and Diversity-Aware CAS** (**HDA-CAS**s) [1] additionally add the *heterogeneity* to the founding principles. This means that they inherently support communication and collaboration among different types of computing elements, such as software, people and sensors. In this paper, we focus on one particular HDA-CAS representative—the **SmartSociety Platform** [3]; and present the design of the platform's *virtualization and communication middleware*—**SmartCom**, providing the communication primitives to supporting the platform's heterogeneity, collectivity and adaptiveness requirements. However, the middleware's design is generally applicable to a wide number of HDA-CAS platforms.

In the following section we present the context and usage scenario of the SmartSociety platform and formulate the design requirements for the SmartCom middleware. In Section 3 we present SmartCom's architecture and functionalities aiming to fulfill the presented requirements. In Section 4 we validate the presented design by describing a prototype implementation and showcase its functioning on a realistic use-case. Section 5 presents the related work. Section 6 concludes the paper.

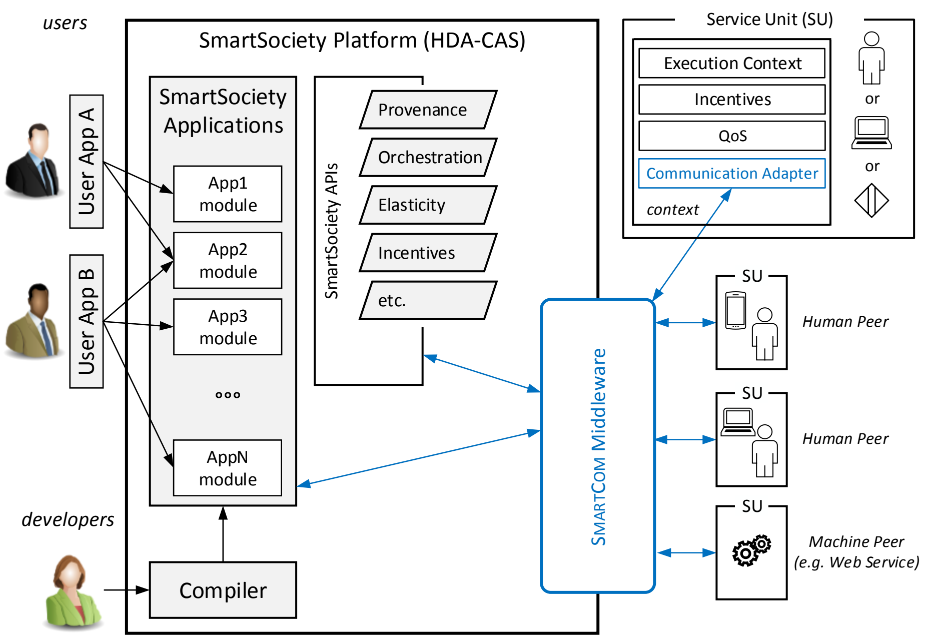

Fig. 1. Operational context for the SmartCom middleware. Middleware components are marked in blue.

2 Motivation & Requirements

2.1 Operational Context

Figure 1 shows the high-level architecture of the SmartSociety platform and presents the SmartCom's operational context. SmartSociety platform supports 'programming' and execution of hybrid human-computer 'computations'. These computations consist of different general-purpose tasks being submitted to the platform for execution. More precisely, the platform *users* submit complex tasks to a *SmartSociety application* running on the platform. The application performs the task by assembling and engaging **collectives** of **service units** to execute the (sub-)tasks collaboratively.

The service unit (SU) is an entity provisioned and utilized through service models (e.g., on-demand and pay-per-use usage) [19]. An SU consists of: *i)* **Peer** — a human, a machine, or a thing (e.g., a sensor) performing the computation or executing a task; *ii)* **Context** — a set of parameters describing the execution context of the particular HDA-CAS application in which the SU is participating. The context parameters can include: execution ID, QoS requirements, performance metrics, associated incentives. Additionally, the context includes a *Communication Adapter*, a component belonging to the SmartCom middleware, providing a context/application-dependent communication and virtualization channel for communicating with the SU, i.e., the peer within it (Figure 1). The SU can use different communication channels to interact with SmartCom, e.g., a human-based SU can communicate with the platform via email and Twitter interchangeably, receive task descriptions and track progress through a web application, and communicate with other SUs within the collective through a dedicated mobile app. Human-based SUs can make use of software-based SUs in the collective, serving as collaborative and utility tools. For example, a software service like Doodle can be used to agree upon participation times, Dropbox as a common repository for performed tasks, or Splunk for data analytics.

Both humans and machines can drive the task processing—e.g., a software may invoke workflow activities which are performed by human-based service units; or, conversely, human-based service units can orchestrate the execution independently, using software services as data analytics, collaboration and coordination tools. How exactly a task is processed is effectively controlled by the SmartSociety application. As an important design principle of the SmartSociety platform is to achieve 'smartness' by leveraging human intelligence and capabilities whenever possible, the applications try to minimize the use of conventional workflows to describe the task processing, and rely primarily on the use of *collaboration patterns*. A collaboration pattern governs the effort within a collective in a loose manner; rather than over-regulating humans, the collaboration patterns set the limits within which the service units are allowed to self-organize, using familiar collaboration tools and environments. A collaborative pattern consists of the following elements:

- Relationship topology – specifying different topologies (e.g., independent, tree, ring, sink, random) and relation types formalizing relationships among service units in a collective. The meaning of the relations is application specific, and can be used to express communication flow, data flow or command flow.
- Collaboration environment – specifying access to familiar external tools that the service units can use to collaborate among themselves (e.g., Google Docs, Dropbox). When a collective is formed, service units are provided with instructions and appropriate access credentials for the previously set up collaboration environment.
- Communication channels – analogously to the collaboration environment, the pattern should specify access to familiar external tools that the service units can use to communicate among themselves and with SSP.

– Elasticity policies – definitions of metrics to be monitored and algorithms for collective composition and adaptation to keep metric values within required limits.
– Security and privacy policies – policies to restrict communication to specific (sub-)collective or to predefined communication channels.

2.2 Motivating Scenario

Let us consider a smart-city *maintenance provider* (*MP*)— a company running a monitoring center covering thousands of sensors and equipment systems geographically dispersed in numerous smart buildings (e.g., Galaxy [4]). The MP provides the centralized service of both *predictive* and *corrective* maintenance to its customers (building owners/tenants). This means that the center actively monitors events originating from various sensors, performs Complex Event Processing (CEP) on these data flows, and if a potential or actual malfunction is detected dispatches collectives of experts to analyze the situation in detail, and, if necessary, perform the physical maintenance work on the ground. The experts are contracted to work on-demand with the MP, subject to their availability. Since each equipment manufacturer defines different issue analysis and reparation procedures, when equipment from different manufacturers is interconnected in a smart building, detecting a cause of an anomaly event sequence cannot easily be done by following prescribed procedures. The complexity grows further when considered at the scale of a smart city, with thousands of building, each with a unique equipment mix, age, environment, and agreed service-level. Therefore, such a scenario does not lend itself well to a workflow type of orchestration. Rather, collectives of human experts perform loosely-controlled collaboration patterns (Section 2.1) in order to detect and repair the problem in the most efficient way, considering the particular context, and making use of supporting software tools when needed (e.g., for data analysis, communication).

2.3 Design Requirements

Based on the previously described desirable characteristics of HDA-CASs and the intended usage context of the SmartSociety platform we can formulate the following SmartCom design requirements:

1. **Virtualization**—supporting heterogeneous human and machine service units as uniformly addressable entities; Supporting 'collective' as a first-class, dynamically-defined entity.
2. **Heterogeneity**—supporting various types of communications channels (protocols) between the platform and service units as well as among service units/collectives, transparently to the platform user.
3. **Communication**—providing primitives for: message transformation, routing, delivery with configurable options (e.g., retries, expiry, delayed, acknowledged).
4. **Persistence**—message persistence and querying.

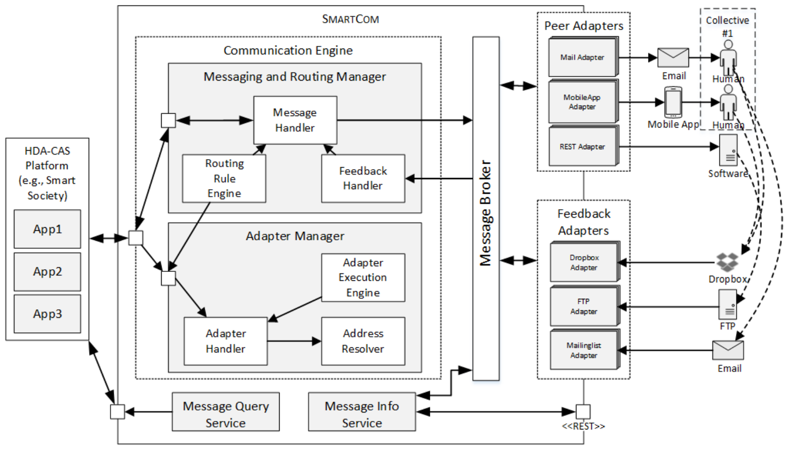

Fig. 2. Simplified architecture of the SmartCom middleware.

5. **Scalability**—ability to handle large number of intermittently available service units.

3 Middleware Design and Architecture

Figure 2 shows the conceptual architecture of the SmartCom middleware. The *HDA-CAS Platform* components (e.g., executing application modules) pass the messages intended for collectives to the *Communication Engine* through a public API. The task of the Communication Engine is to effectively virtualize the notions of 'collective' and 'service unit (SU)' for the HDA-CAS platform. This means that the communication with different service units and collectives has to be handled transparently to the HDA-CAS platform, independent of unit's actual type and availability. In the following sections, for brevity, when referring to the communicational aspect of SU's functionality we will use the short term "peer" denoting the computing element within the SU that is the sender or receiver of information/data; and the term "adapter" denoting the middleware component in charge of handling the communication.

3.1 Messaging and Routing

All communication between the peers and the platform is handled asynchronously using messages. A queue-based *Message Broker* is used to decouple the execution of SmartCom's components and the communication with peers. SmartCom supports unicast as well as multicast messages. Therefore peers can be addressed as collectives and the SmartCom will take care of sending the message to every single member of the collective.

The *Messaging and Routing Manager* (MRM) is SmartCom's principal entry point for HDA-CASs. It consists of the following components: 1. The *Message Handler* takes incoming messages from HDA-CAS and transforms them into an internal representation. If the receiver of the message is a collective, it resolves the current member peers, and their preferred communication channels; 2. The *Routing Rule Engine* then determines the proper route to the peers, invoking the Adapter Manager to instantiate appropriate adapters in order to complete the route, if needed (see below); 3. The *Feedback Handler* waits for feedback messages received through feedback adapters and passes them to the Message Handler. Afterwards they will be handled like normal messages again, and re-routed where needed, e.g., back to the HDA-CAS.

A *route* may include different communication channels as delivery start/endpoints. Figure 3 shows the conceptual overview of SmartCom's routing. For each message the route will be determined by the Routing Rule Engine using the pipes-and-filters pattern, determining the route based on the message properties: receiver ID, message type and message subtype, with decreasing priority. Note that there may be multiple routes per message (e.g., a single peer can be contacted using a mobile application and/or SMS, or access the system via a mobile app and web browser concurrently).

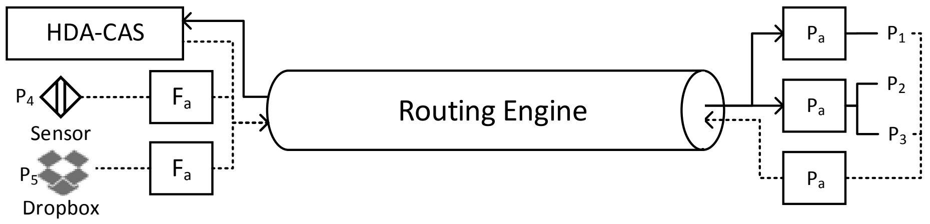

Fig. 3. Messages are routed to Peer Adapters (P_a) which forward the messages to the corresponding Peers (P_1 to P_5). Feedback is sent back by human peers, software peers (e.g., Dropbox) and sensors using Feedback Adapters (F_a). The HDA-CAS Platform can also send and receive messages.

3.2 Adapters

In order to use a specific communication channel, an associated *adapter* needs to be instantiated. The communication between peers and the adapters is unidirectional—*peer adapters* are used to send messages to the peers; *feedback adapters* are used to receive messages from peers. SmartCom originally provides some common peer/feedback adapters (e.g., SMTP/POP, Dropbox, Twitter). In addition, being developed in the context of a research project, it also provides adapters for dedicated SmartSociety mobile/web peer apps. When an HDA-CAS requires the SmartCom to handle a new communication channel, it needs to provide an appropriate adapter implementation. Afterwards, SmartCom will instantiate the new adapter when needed.

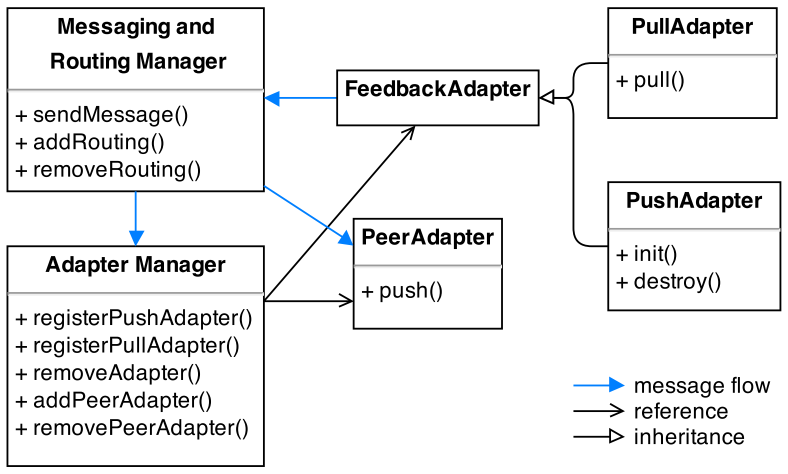

Fig. 4. Partial representations of the SmartCom's APIs and message flow.

The role of adapters should be considered from the following two perspectives: *i)* functional; and *ii)* technical.

Functionally, the adapters allow for: *a)* Hybridity — by enabling different communication channels to and from peers; *b)* Scalability — by enabling SmartCom to cater the dynamically changing number of peers; *c)* Extensibility — new types of communication and collaboration channels can easily be added at a later stage transparently to the rest of the HDA-CAS platform. *d)* Usability — human peers are not forced to use dedicated applications for collaboration, but rather freely communicate and (self-)organize among themselves by relying on familiar third-party tools. *e)* Load Reduction and Resilience — by requiring that all the feedback goes exclusively and unidirectionally through external tools first, only to be channelled/filtered later through a dedicated feedback adapter, the SmartCom is effectively shielded from unwanted traffic load, delegating the initial traffic impact to the infrastructure of the external tools. At the same time, failure of a single adapter will not affect the overall functioning of the middleware.

Technically, the primary role of adapters is to perform the message format transformation. Optional functionalities include: message filtering, aggregation, encryption, acknowledging and delayed delivery. Similarly, the adapters are used to interface SmartCom with external software services, allowing the virtualization on third party tools as common software peers.

The *Adapter Manager* is the component responsible for managing the adapter lifecycle (i.e., creation, execution and deletion of instances), elastically adjusting the number of active instances from a pool of available adapters. This allows the SmartCom to scale the number of active connections up and down as needed. This is especially important when dealing with human peers, due to their inherent periodicity, frequent instability and unavailability, as well as for managing a large number of connected devices, such as sensors. The Adapter Manager consists of following subcomponents:

- *Adapter Handler*: managing adapter instance lifecycle. It handles the following adapter types:
  - Stateful peer adapters— peer adapters that maintain conversation state (e.g., login information). For each peer a new instance of the adapter will be created and handled by the Adapter Manager;
  - Stateless peer adapters— peer adapters that maintain no state. An instance of an adapter can send messages to multiple peers;
  - Feedback pull adapters— adapters that actively poll software peers for feedback. They are created on demand by applications running on the HDA-CAS platform and will check regularly for feedback on a given communication channel (e.g., check if a file is present on an FTP server);
  - Feedback push adapters— adapters that wait for feedback from human/machine peers.
- *Adapter Execution Engine*: executing the active adapters.
- *Address Resolver*: mapping adapter instances with peers' external identifiers (e.g., Skype/Twitter username) in order to initiate the communication.

Feedback messages from peers (e.g., subtask results) or external tools (e.g., Dropbox file added, email received on a mailing list) are consumed by the adapters either by a push notification or by pulling in regular intervals (more details in Section 4).

Figure 4 shows the simplified APIs of the previously described components, as well as the message flow among SmartCom's components. Due to space constraints, a detailed description of the described architectural components and their implementation, as well as the full API specification is provided in the supplement materials[1].

3.3 Other Functionalities

All sent and received messages as well as internal messages are persisted in a NoSQL database. Stored messages can be queried and analyzed through the *MessageQuery* public API (e.g., to derive metrics or identify conditions for applying incentives). Since messages can be of arbitrary subtype and contain an arbitrary payload, human peers (and their local third-party applications) might not know how to interpret the message. The *MessageInfoService* provides: *a)* The semantic meaning/description of message type and contents in a human-readable way; *b)* Dependencies to other messages; *c)* Timing constraints (e.g., expiry, priority). This is especially useful when supporting complex task acceptance negotiations, where human peers are required to fully understand the message meaning and send back valid answers. Currently, the service annotates the message field types,

[1] https://github.com/tuwiendsg/SmartCom/wiki

provides a natural-language description of the expected fields contents and provides a state-machine description describing the allowed message exchange sequence with respect to dependency and timing constraints. The service can also be extended to provide an ontology of message types enabling machine-readable message descriptions, and use of personal software agents searching for tasks and participating in negotiations on behalf of human peers [20].

4 Implementation & Illustrative Example

SmartCom prototype was implemented in Java and can be used directly by HDA-CAS platforms running on the Java Virtual Machine. Additionally, other platforms can interact with SmartCom using the set of provided APIs[1]. The prototype comes with some standard adapters (e.g., Email, Twitter, Dropbox) implemented. Additional third-party adapters can be loaded as plug-ins and instantiated when needed. SmartCom uses MongoDB[2] as a database system for its various subsystems. Depending on the usage of the middleware, either an in-memory or dedicated database instances of MongoDB can be used. To decouple the execution of the HDA-CAS platform and the communication we use Apache ActiveMQ[3] as the message broker. Additional information on the implementation, usage and the source code can be found on GitHub[4].

4.1 Use-Case Walkthrough

Based on the motivating scenario presented in Section 2.2 we will now formulate a concrete use-case to validate the presented design and its fulfillment of the stated requirements:

A predictive maintenance SmartSociety application receives sensor readings from a smart building and performs Complex Event Processing (CEP) on them. If an indication for a potential malfunction is detected, further investigation is required. A collective (COL1) of available human experts is formed[5] *and a collaborative pattern imposed (Section 2.1). The pattern appoints a leading expert to manage the peer collaboration within the collective and sets up a Dropbox repository for sharing the findings and equipment logs between the SmartSociety application and the collective. Additionally, it provides to the COL1 manager the contact details of the manufacturer of the malfunctioning equipments in case additional consultations are required. Finally,* SmartCom *also provides COL1 peers with mediated access to a data analysis tool (e.g., Splunk*[6]*).*

[2] `http://www.mongodb.org`
[3] `http://activemq.apache.org`
[4] https://github.com/tuwiendsg/SmartCom
[5] Selection of collective peers is out of scope of this paper.
[6] `www.splunk.com`

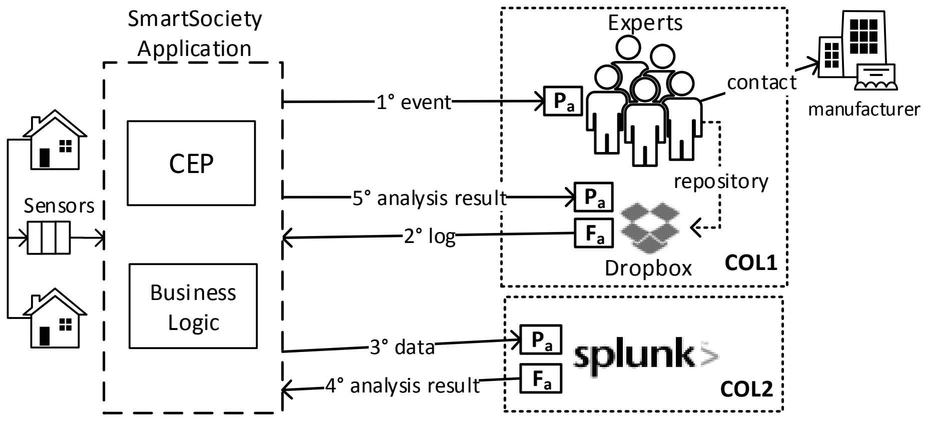

Fig. 5. Supporting predictive maintenance use-case. Collectives of human expert and software service units participate in a joint collaboration to identify the cause of a detected malfuntion event.

Figure 5 shows the two collectives participating in this scenario. COL1 containing human expert servise units (SUs) and a single software SU — the Dropbox service. Furthermore, each human SU is assigned a dedicated peer adapter (P_a) instance, while for the Dropbox service, both a P_a and a feedback adapter (F_a) instance are executed, in order to support two-way communication with the SmartSociety platform. COL2 contains a single SU that does data analysis. To support two-way communication we introduce again a P_a and a F_a.

The use-case starts by SmartSociety application notifying peers that their participation is needed (Fig. 5, 1°). Some peers expressed in their profiles the preference for being notified by SMS, others by email. To send an SMS SmartCom reads the phone number of a peer from its profile and hands it to the stateless SMS adapter. The adapter sends the message using the most cost-efficient mobile operator. Those peers that prefer to be contacted through email will be sent an email using a stateless email adapter through an external mail service. The contents of the message are provided by the SmartSociety application. In this case, the message contains the URL pointing to the description of the detected event, Dropbox repository URL and access tokens for sharing the results, the name and contact details of the selected collective manager as well as a natural language description of the required activities and contractual terms. Furthermore, the manager is sent the contact details of the equipment manufacturer's customer service, and the address of another collective – COL2, which in practice contains a single software peer, the Splunk service.

For the sake of simplicity, we assume that expert peers do accept the terms and participate in COL1. The manager freely organizes the collaboration in COL1. Human peers examine the logs. At a certain point, they need to run an additional data analysis on the log. The collaboration pattern foresees that if a file with predefined filename is deposited in the shared Dropbox repository, the dedicated feedback adapter would pick up that file (2°) and forward it to the COL2 for analysis. The middleware ensures that the feedback pull adapter for Dropbox regularly checks if there are new files available (e.g., once a minute). The system will then create and send a message to the Splunk Peer Adapter which contains the location of the file and further information on the analysis (3°). Once Splunk has finished analyzing the data, Splunk will deposit the results file back to the Dropbox repository (4° + 5°) and its feedback push adapter will push a multicast notification message to the COL1 members (1° again). The COL1 can then continue their work.

5 Related Work

The research field of HDA-CAS is new, but highly varied, and in its many niches builds upon different founding technologies and research results, spanning research areas, such as: autonomous agents, robotics, bio-inspired collectives, human-provided services, crowdsourcing and web-scale workflow technologies. *Multi-agent based systems* focus on the peer-to-peer communication between agents, which are usually considered as software entities. The main focus lies on interaction between those agents which act independently of each other. Approaches on how to handle communication in MAS have been made by the "Foundation for Intelligent Physical Agents" [6] (FIPA), which were used for example in the Java Agent DEvelopment (JADE) Framework [7]. Another popular language is the KQML [8] (Knowledge Query and Manipulation Language). Our approach, on the other hand, also incorporates humans as first-class actors and focuses on the interaction of the humans with the system. In [5], authors propose a middleware that supports communication among agents on different platforms and programming languages. They use a different runtime for each platform and exchange messages between those runtimes to achieve a cross-platform communication of agents. Compared to our approach it focuses on the peer-to-peer interaction instead of the interaction of the system with peers. The intention of the middleware is to exchange the messages between the runtime systems compared to direct message exchange with peers in our approach. In *Swarm Robotics* the work on peer communication is multifaceted. In the ASCENS project [10] one of the communication strategies relies on the visual communication [9]. The coordination is decentralised, contrarily to our approach, which requires a centralised component that coordinates the communication. Some swarm robotic projects/algorithms completely avoid communication among peers (e.g., [11]). *Service-based approaches* usually involve addressing peers as Web Services. For example the ALLOW Ensembles project [12] concentrates on the concept of cell ensembles, which consist of cells that have a defined behaviour. They use BPEL4Chor [13] for the communication between cells, which allows communication between web services. The ASCENS project focuses on the peer-to-peer approach where some peers of the system know at least some other peers in the system [14]. They use Pastry [15] and extend it with the SCRIBE protocol [16] to support any- and multi-casts. As previously shown, this behavior differs from ours, in that we do not support anycast, but multicast specifically within collective boundaries. *Social Computing platforms* like Jabberwocky [17] or TurKit [18] utilize human capabilities to solve problems. However, they rely on existing crowdsourcing platforms and adopt their model of human peers, not supporting hybrid machine/human collectives.

6 Conclusions and Future Work

In this paper we presented SmartCom — a middleware for SmartSociety platform, a typical HDA-CAS. SmartCom aims to tackle hybridity in a variety of aspects, including different kinds of supported computations (human- vs. machine-based service units), different channels of communication, and loose-coupling to promote use of familiar third-party services. This heterogeneity is of high importance in order to create a platform, which is able to scale to a potentially high number of service units organized in multiple dynamic collectives. SmartCom allows addressing collectives of service units transparently to the HDA-CAS, relieving the HDA-CAS programmer the duty to keep track of current members of a collective, allowing the collective to scale up and down when needed seamlessly. The described design was chosen to fulfill the outlined requirements, and was validated through a prototype implementation.

The focus of our future research will be on modelling the primitives for integrated monitoring and execution of elasticity actions, such as imposing of optimal topologies, dynamical adjustment of collective members, and support for incentive application. Currently, these actions have to be fully specified on the HDA-CAS (SmartSociety) application level, presenting an unnecessary burden for the developers.

Acknowledgment

This work is supported by the EU FP7 SmartSociety project under grant № 600854.

References

1. F. Giunchiglia and V. Maltese and S. Anderson and D. Miorandi. *Towards Hybrid and Diversity-Aware Collective Adaptive Systems*, First FOCAS Workshop on Fundamentals of Collective Systems @ECAL, 2013. http://eprints.biblio.unitn.it/4214/
2. S. Anderson et al. *FoCAS Book: Adaptive Collective Systems Herding Black Sheep*, Open publication, ISBN pending, 2013. http://focas.eu/documents/adaptive-collective-systems.pdf
3. D. Miorandi and V. Maltese and M. Rovatsos and A. Nijholt and J. Stewart. *Social Collective Intelligence: Combining the Powers of Humans and Machines to Build a Smarter Society*, ISBN 978-3-319-08680-4, Springer, 2014.
4. Pacific Contorls Galaxy. http://www.pacific-galaxy.com/

5. G. Cabri, E. Domnori and D. Orlandini. *Implementing Agent Interoperability between Language-Heterogeneous Platforms*, Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2011 20th IEEE International Workshops on , vol., no., pp.29,34, 27-29 June 2011
6. Foundation for intelligent physical agents. http://www.fipa.org/. Accessed: 2014-07-09.
7. Java agent development framework. http://jade.tilab.com/. Accessed: 2014-05-28.
8. T. Finin, R. Fritzson, D. McKay, and R. McEntire. *Kqml as an agent communication language.* In Proceedings of the third international conference on Information and knowledge management, pages 456463. ACM, 1994.
9. N. Capodieci and G. Cabri. *Collaboration in swarm robotics: A visual communication approach*, Collaboration Technologies and Systems (CTS), 2013 International Conference on , vol., no., pp.195,202, 20-24 May 2013.
10. F. Zambonelli, N. Bicocchi, G. Cabri, L. Leonardi, and M. Puviani. 2011. *On Self-Adaptation, Self-Expression, and Self-Awareness in Autonomic Service Component Ensembles.* In Proceedings of the 2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW '11). IEEE Computer Society, Washington, DC, USA, 108-113.
11. D. Kengyel, R. Thenius, K. Crailsheim, and T. Schmickl: *Influence of a Social Gradient on a Swarm of Agents Controlled by the BEECLUST Algorithm.* Proceedings of the 12th European Conference on Artificial Life (2013), 1041-1048.
12. A. V. Andrikopoulos, A. Bucchiarone, S. Gomez Saez, D. Karastoyanova, and C. Antares Mezzina. *Towards Modelling and Execution of Collective Adaptive Systems.* 9th International Workshop on Engineering Service-Oriented Applications WESOA 2013), In conjunction with ICSOC 2013, December 2nd 2013, Berlin, Germany.
13. G. Decker; O. Kopp; F. Leymann; M. Weske, *BPEL4Chor: Extending BPEL for Modeling Choreographies*, Web Services, 2007. ICWS 2007. IEEE International Conference on , vol., no., pp.296,303, 9-13 July 2007.
14. P. Mayer, A. Klarl, R. Hennicker, M. Puviani, F. Tiezzi, R. Pugliese, J. Keznikl, and T. Bures. *The Autonomic Cloud: A Vision of Voluntary, Peer-2-Peer Cloud Computing.* In Proceedings of 3rd Workshop on Challenges for Achieving Self-Awareness in Autonomic Systems, pages 16, Philadelphia, USA, September 2013.
15. A. I. T. Rowstron and P. Druschel, *Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems*, in Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, ser. Middleware 01. London, UK, UK: Springer-Verlag, 2001, pp. 329350.
16. M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. Rowstron, *Scribe: A large-scale and decentralized application-level multicast infrastructure*, Selected Areas in Communications, IEEE Journal on, vol. 20, no. 8, pp. 14891499, 2002.
17. S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar. 2011. *The jabberwocky programming environment for structured social computing.* In Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST '11). ACM, New York, NY, USA, 53-64. DOI=10.1145/2047196.2047203 http://doi.acm.org/10.1145/2047196.2047203
18. G. Little, L. B. Chilton, M. Goldman, and R. C. Miller. *Turkit: tools for iterative tasks on mechanical turk.* In Proceedings of the ACM SIGKDD workshop on human computation, pages 2930. ACM, 2009.
19. H.-L. Truong, S. Dustdar, and K. Bhattacharya. *Conceptualizing and Programming Hybrid Services in the Could.* Int. J. Coop. Info. Syst. 22, 1341003 (2013) DOI: 10.1142/S0218843013410037

20. Y. Gal, S. Kraus, M. Gelfand, H. Khashan, and E. Salmon. *An Adaptive Agent for Negotiating with People in Different Cultures*. In ACM Trans. Intell. Syst. Technol. vol. 3, no. 1, ACM, 2011. DOI: 10.1145/2036264.2036272