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, Vienna, Austria
{zeppezauer,oscekic,truong,dustdar}@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. The flexibility of these collectives makes them suitable for processing elaborate tasks, but at the same time, building a system to support diverse communication types in 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 of diverse types of service units. The middleware is able to handle numerous, intermittently available, human and software-based service units, and manages the notion of collectivity transparently to the programmer. A prototype implementation for validation purpose is also provided.

1 Introduction

Collective Adaptive System (*CAS*) [1] is an umbrella-term denoting distributed systems comprised of multiple autonomous computing/service units with different individual properties, but with the fundamental property of collectiveness. *Collectiveness* implies that the individual units 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 units may join and leave, and dynamically alter collective compositions or task execution goals. CASs come in a variety of forms. *Hybrid and Diversity-Aware CASs* (*HDA-CASs*) [2] additionally add the *heterogeneity* to the founding principles. This means that they inherently support communication and collaboration among different types of units, such as software, people and sensors.

Motivation. 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[1]).

[1] Pacific Controls Galaxy. http://www.pacific-galaxy.com/.

F. Toumani et al. (Eds.): ICSOC 2014, LNCS 8954, pp. 56–67, 2015.
DOI: 10.1007/978-3-319-22885-3_6

The MP provides the centralized service of both *predictive* and *corrective* maintenance to its customers (building/equipment owners/tenants). This means that MP control centers actively monitor events originating from various sensors and perform Complex Event Processing on these data flows. If a potential or actual malfunction is detected they dispatch collectives of experts to analyze the situation in detail, and, if necessary, perform the physical maintenance work on the ground. The (human) 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 the 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 conventional workflow type of orchestration. Rather, collectives of human experts perform loosely-controlled collaboration patterns (Sect. 2) 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).

Contribution. In the described motivational scenario the MP needs a platform to communicate with, deploy and orchestrate ad-hoc assembled, dynamic teams of human-based and machine-based service units in order to execute, often unpredictable, complex collaboration patterns. A HDA-CAS, as the one being researched in *SmartSociety* project [3], can support these requirements. In this paper, we present the design of one of the SmartSociety's core components: the *virtualization and communication middleware* – SmartCom, providing the communication and virtualization primitives to support heterogeneity, collectivity and adaptiveness. SmartCom is actually an independent component usable with a wide number of HDA-CAS platforms. SmartCom fulfils the following characteristic of most service buses: (a) *Heterogeneity* – supporting various types of communications channels (protocols) between the platform and service units as well as among service units/collectives, transparently to the HDA-CAS platform user. (b) *Communication* – providing primitives for: message transformation, routing, delivery with configurable options (e.g., retries, expiry, delayed, acknowledged). (c) *Persistence* – message persistence and querying. (d) *Security* – Handling authentication and encryption, as well as preventing message flooding. (e) *Scalability* – ability to handle large number of intermittently available service units.

In addition to these features, the distinguishing novelty of SmartCom is its native support for virtualizing collectives: (i) SmartCom hides the complexity of communication with a dynamic collective as a whole and passing of instructions from the HDA-CAS execution engine to it, making it a first-class, programmable entity; (ii) Communication with the collective members is transparent to the HDA-CAS, regardless of whether they are human or machine-based, with SmartCom interpreting the messages for the human-based service units; (iii) A single human, sensor or software service endpoint can participate in different

collectives concurrently, acting as a different service unit with different SLA, delivery and privacy policies. These novel properties make SmartCom especially suitable for supporting scenarios such as predictive maintenance. To the best of our knowledge, no other platforms or middleware systems offer the collective virtualization in a similar manner.

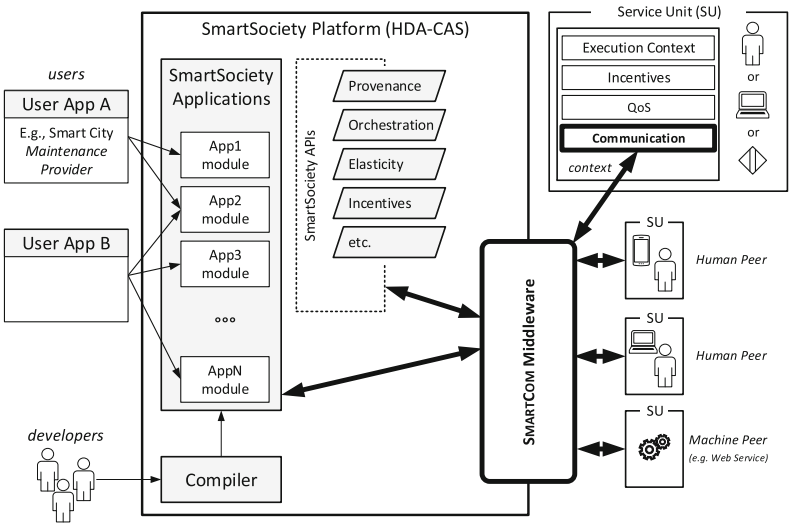

Fig. 1. Operational context for the SmartCom middleware. Middleware components are marked with thick lines.

Paper Organization. In the following section we present the operational context of the SmartCom middleware. In Sect. 3 we present SmartCom's architecture and design choices. In Sect. 4 we describe the implementation and illustrates a realistic use-case. Section 5 presents the related work. Section 6 concludes the paper.

2 Operational Context of Collective Communication

Figure 1 shows the high-level architecture of the SmartSociety platform and presents 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* (e.g., a smart-city maintenance provider) 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 use), as described in [4]. 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. To describe the communication with a SU, the context comprises the *Communication Context* part which defines the context-dependent communication and virtualization channels (e.g., using email, SMS). 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 collaboration pattern consists of the following elements: (1) *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/data/command flow. (2) *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. (3) *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 SmartSociety Platform. (4) *Elasticity policies* – definitions of metrics to be monitored and algorithms for collective composition and adaptation. (5) *Security and privacy policies* – policies to restrict communication to specific (sub-)collective or to predefined communication channels.

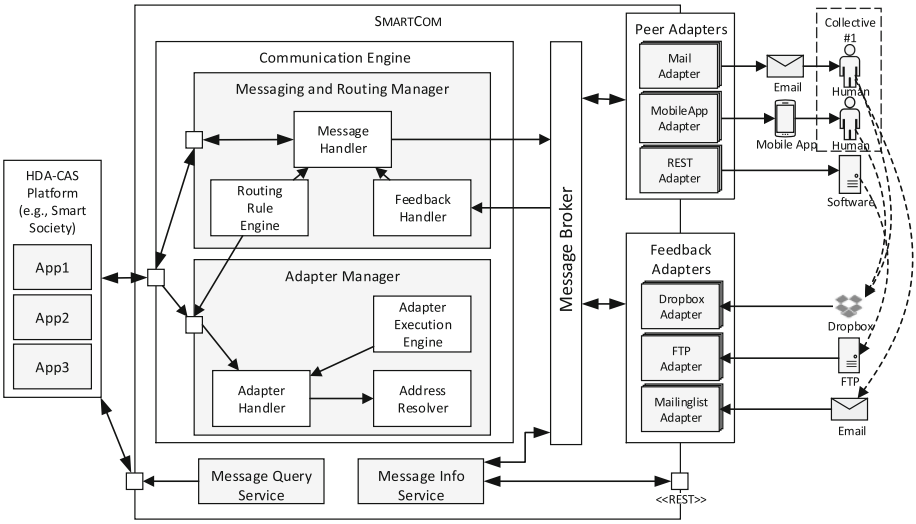

Fig. 2. Simplified architecture of the SmartCom middleware.

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 human/machine 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.

Messaging and Routing. All communication between the peers and the platform is handled asynchronously using normalized 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, multiple peers can also 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 app, email and SMS concurrently).

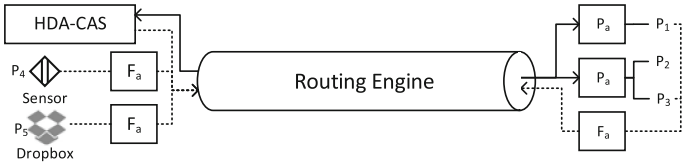

Fig. 3. Messages are routed to Peer Adapters (P_a) which forward the messages to the corresponding Peers (P_1 to P_5). Feedback is sent back by human peers, software peers (e.g., Dropbox) and sensors using Feedback Adapters (F_a). The HDA-CAS Platform can also send and receive messages.

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 Android/Web peer apps. The role of adapters should be considered from the following two perspectives: *(1)* functional; and *(2)* technical.

Functionally, the adapters allow for: *(a)* Hybridity – by enabling different communication channels to and from peers; *(b)* Scalability – by enabling SmartCom to cater to 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 scaling the number of active adapter instances out 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: *(i)* 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; *(ii)* Stateless peer adapters – peer adapters that maintain no state. An instance of an adapter can send messages to multiple peers; *(iii)* 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); *(iv)* Feedback push adapters – adapters that wait for feedback from 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 Sect. 4). 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[2].

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

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

service annotates the message field types, 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 *MessageInfoService* 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 [5].

SmartCom supports specifying and observing delivery and privacy policies on message, peer and collective level: *Delivery policies* stipulate how to interpret and react to possible communication exceptions, such as: failed, timed out, unacknowledged or repeated delivery. *Privacy policies* restrict sending or receiving messages or private data to/from other peers, collectives or HDA-CAS applications under different circumstances. Apart from offering predefined policies, SmartCom also allows the users to import custom, application- or peer-specific policies. As noted, both types of policies can be specified at different levels. For example, a peer may specify that he can be reached only by peer 'manager' via communication channel 'email', from 9am to 5pm in collective 'Work'. The same person can set to be reachable via 'SMS' any time by all collective members except 'manager' in collective 'Bowling'. Similarly, a HDA-CAS platform application could specify the collective delivery policy stating that when sending instructions to a collective it suffices that the delivery to a single member succeeds to consider the overall delivery successful on the collective level. SmartCom takes care of combining and enforcing these policies transparently to the HDA-CAS user in different collective contexts.

Peer authentication is handled externally. Before instantiating the corresponding adapter, SmartCom requires the peers to authenticate with the external tool and obtains from the tool the token that is used to authenticate messages from/to the peer. More information is provided in the supplement materials.

4 Implementation and Illustrative Example

SmartCom prototype was implemented in the Java programming language 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. The prototype comes with some implemented standard adapters (e.g., Email, Twitter, Dropbox) that can be used to test, evaluate and operate the system. Additional third-party adapters can be loaded as plug-ins and instantiated when needed. SmartCom uses MongoDB[3] 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[4] as the message broker. The source code, as well as runnable integration tests showcasing the usage and functioning of the middleware can

[3] http://www.mongodb.org.
[4] http://activemq.apache.org.

be found in SmartCom's GitHub repository[5]. The various subsystems and the whole system can be built using Maven. The APIs are provided in the `api` module. Additional documentation regarding the design, implementation and usage is provided on the repository's Wiki page.

Based on the motivating scenario presented in Sect. 1 we formulate a concrete use-case to validate the presented design and its fulfilment 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 of a potential malfunction is detected, further investigation is required. A collective (COL1) of available human experts is formed*[6] *and a collaborative pattern imposed (Sect. 2). The application appoints an expert to lead 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*[7]*).*

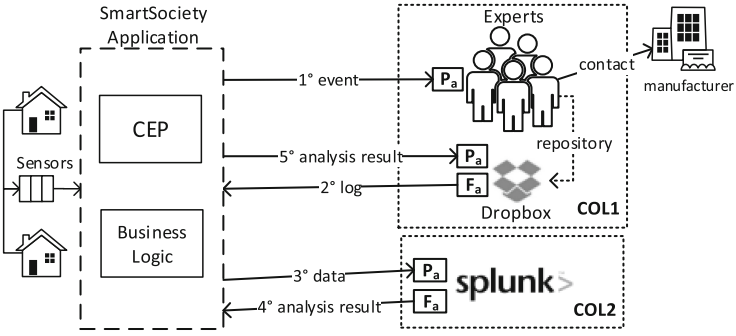

Fig. 4. Supporting predictive maintenance use-case. Collectives of human expert and software service units participate in a joint collaboration to identify the cause of a detected malfunction event.

Figure 4 shows the two collectives participating in this scenario. COL1 containing human expert service 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. 4, 1°) by sending a message to `MessagingAnd`

[5] https://github.com/tuwiendsg/SmartCom.
[6] Selection of collective peers is out of scope of this paper.
[7] www.splunk.com.

`RoutingManager` which will initialize the routing. Some peers expressed in their profiles the preference for being notified by SMS, others by email. To send an SMS `MessagingAndRoutingManager` reads the phone number of a peer from its profile and hands it to `AdapterManager` which instantiates and executes the SMS adapter. `PeerAdapter` sends the message by 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. This preference can be set using `DeliveryPolicy`. 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. At a certain point, human peers 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 `FeedbackPullAdapter` for Dropbox `(DropboxFeedbackAdapter)` 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 `FeedbackPushAdapter` will push a multicast notification message to the COL1 members (1° again). The COL1 can then continue their work.

5 Related Work

SmartCom encompasses different design choices that, taken individually, can be compared with existing solutions. However, to the best of our knowledge, no existing system incorporates a similar set of functionalities as the one SmartCom offers to support an effective virtualization of communication for dynamic, hybrid human-machine collectives. Popular open-source and proprietary Enterprise Service Buses and Integration Technologies provide the same support and flexibility for custom adapters as SmartCom does. On the other hand, many ESBs lack the support of multi-tenancy (e.g., Apache ServiceMix[8] and JBossESB[9]) or do have restrictions on implementing custom adapters (e.g., JBossESB). Others do not support the dynamical enforcement of policies (e.g., WSO2

[8] http://servicemix.apache.org.
[9] http://jbossesb.jboss.org/.

ESB[10]) and there is in general no support of the addressing of collectives at all which is one of the key features of SmartCom. Furthermore the support of humans interacting with the system is generally not considered.

Service-oriented CASs usually involve addressing peers as Web Services, lacking the ability to communicate with peers using different communication channels, especially external tools. For example, the ALLOW Ensembles project [6] concentrates on the concept of cell ensembles, which consist of cells that have a defined behaviour. They use BPEL4Chor [7] 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 [8]. They use Pastry [9] and extend it with the SCRIBE protocol [10] 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. In [11], 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. Social Computing platforms like Jabberwocky [12] or TurKit [13] utilize human capabilities to solve problems. However, they rely on existing crowdsourcing platforms and rely on their communication model that does not supporting collectives at all.

6 Conclusions and Future Work

In this paper we presented SmartCom – a middleware supporting hybridity in a variety of aspects, including: different computation types (human- vs. machine-based service units), different communication channels, and loose-coupling to promote use of familiar third-party services. This 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 validated through a prototype implementation.

The focus of our future research will be on modeling 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.

[10] http://wso2.com/.

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

References

1. Anderson, S., Bredeche, N., Eiben, A.E., Kampis, G., van Steen, M.: Adaptive collective systems herding black sheep. In: BookSprints for ICT Research (2013)
2. Giunchiglia, F., Maltese, V., Anderson, S., Miorandi, D.: Towards hybrid and diversity-aware collective adaptive systems. Technical report, Univ. of Trento (2013). http://eprints.biblio.unitn.it/4214/
3. Miorandi, D., Maltese, V., Rovatsos, M., Nijholt, A., Stewart, J. (eds.): Social Collective Intelligence: Combining the Powers of Humans and Machines to Build a Smarter Society. Springer, New York (2014)
4. Truong, H.L., Dustdar, S., Bhattacharya, K.: Conceptualizing and programming hybrid services in the cloud. Int. J. Coop. Inf. Syst. **22**(04), 1341003 (2013)
5. Gal, Y., Kraus, S., Gelfand, M., Khashan, H., Salmon, E.: An adaptive agent for negotiating with people in different cultures. ACM Trans. Intell. Syst. Technol. **3**(1), 8:1–8:24 (2011)
6. Andrikopoulos, V., Bucchiarone, A., Gómez Sáez, S., Karastoyanova, D., Mezzina, C.A.: Towards modeling and execution of collective adaptive systems. In: Lomuscio, A.R., Nepal, S., Patrizi, F., Benatallah, B., Brandić, I. (eds.) ICSOC 2013. LNCS, vol. 8377, pp. 69–81. Springer, Heidelberg (2014)
7. Decker, G., Kopp, O., Leymann, F., Weske, M.: Bpel4chor: extending bpel for modeling choreographies. In: IEEE International Conference on Web Services, ICWS 2007, pp. 296–303 (2007)
8. Mayer, P., Klarl, A., Hennicker, R., Puviani, M., Tiezzi, F., Pugliese, R., Keznikl, J., Bures, T.: The autonomic cloud: a vision of voluntary, peer-2-peer cloud computing. In: 2013 IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops (SASOW), pp. 89–94 (2013)
9. Rowstron, A., Druschel, P.: Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
10. Castro, M., Druschel, P., Kermarrec, A.M., Rowstron, A.: Scribe: a large-scale and decentralized application-level multicast infrastructure. IEEE J. Sel. Areas Commun. (JSAC) **20**(8), 1489–1499 (2002)
11. Cabri, G., Domnori, E., Orlandini, D.: Implementing agent interoperability between language-heterogeneous platforms. In: 20th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), pp. 29–34 (2011)
12. 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 (2011)
13. Little, G.: Turkit: Tools for iterative tasks on mechanical turk. In: IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2009, pp. 252–253 (2009)