Leveraging the Service Bus Paradigm for Computer-mediated Social Communication Interoperability

Rafael Angarita, Nikolaos Georgantas, Cristhian Parra, James Holston, Valérie Issarny

Leveraging the Service Bus Paradigm for Computer-mediated Social Communication Interoperability

Rafael Angarita*, Nikolaos Georgantas*, Cristhian Parra*†, James Holston† and Valérie Issarny*
*MiMove Team, Inria Paris, France.
firstname.lastname@inria.fr
† Department of Anthropology
University of California Berkeley, CA, USA
jholston@berkeley.edu

***Abstract*—Computer-mediated communication can be defined as any form of human communication achieved through computer technology. From its beginnings, it has been shaping the way humans interact with each other, and it has influenced many areas of society. There exist a plethora of communication services enabling computer-mediated social communication (e.g., Skype, Facebook Messenger, Telegram, WhatsApp, Twitter, Slack, etc.). Based on personal preferences, users may prefer a communication service rather than another. As a result, users sharing same interests may not be able to interact since they are using incompatible technologies. To tackle this interoperability barrier, we propose the Social Communication Bus, a middleware solution targeted to enable the interaction between heterogeneous communication services. More precisely, the contribution of this paper is threefold: (i), we propose a survey of the various forms of computer-mediated social communication, and we make an analogy with the computing communication paradigms; (ii), we revisit the *eXtensible Service Bus* (XSB) that supports interoperability across computing interaction paradigms to provide a solution for computer-mediated social communication interoperability; and (iii), we present Social-MQ, an implementation of the Social Communication Bus that has been integrated into the AppCivist platform for participatory democracy.**

***Index Terms*—Social Communication, Computer-mediated Communication, Interoperability, Middleware, Service-oriented Architecture.**

I. Introduction

People increasingly rely on computer-mediated communication for their social interactions (e.g., see [1]). This is a direct consequence of the global reach of the Internet combined with the massive adoption of social media and mobile technologies that make it easy for people to view, create and share information within their communities almost anywhere, anytime. The success of social media has further led - and is still leading - to the introduction of a large diversity of social communication services (e.g., Skype, Facebook, Google Plus, Telegram, Instagram, WhatsApp, Twitter, Slack, ...). These services differ according to the types of communities and interactions they primarily aim at supporting. However, existing services are not orthogonal and users ultimately adopt one service rather than another based on their personal experience (e.g., see the impact of age on the use of computer-based communication in [2]). As a result, users who share similar interests from a social perspective may not be able to interact in a computer-mediated social sphere because they adopt different technologies. This is particularly exacerbated by the fact that the latest social media are proprietary services that offer an increasingly rich set of functionalities, and the function of one service does not easily translate -both socially and technically- into the function of another. As an illustration, compare the early and primitive computer-mediated social communication media that is email with the richer social network technology. Protocols associated with the former are rather simple and email communication between any two individuals is now trivial, independent of the mail servers used at both ends. On the other hand, protocols associated with today's social networks involve complex interaction processes, which prevent communication across social networks.

The above issue is no different from the long-standing issue of interoperability in distributed computing systems, which requires mediating (or translating) the protocols run by the interacting parties for them to be able to exchange meaningful messages and coordinate. And, while interoperability in the early days of distributed systems was essentially relying on the definition of standards, the increasing complexity and diversity of networked systems has led to the introduction of various interoperability solutions [3]. In particular, today's solutions allow connecting networked systems in a non-intrusive way, i.e., without requiring to modify the systems [4], [5], [6], [7], [8]. These solutions typically use intermediary software entities whose name differ in the literature, e.g., mediators [9], wrappers [4], mediating adapters [5], or binding components [10]. However, the key role of this software entity, whatever its name, is always the same: it translates the data model and interaction processes of one system into the ones of the other system the former needs to interact with, assuming of course that the systems are functionally compatible. In the following, we use the term *binding component* to refer to the software entity realizing the necessary translation. The binding component is then either implemented in full by the developer,

or synthesized - possibly partially - by a dedicated software tool (e.g., [7]).

The development of binding components depends on the architecture of the overall interoperability system, since the components need to be deployed in the network and connected to the systems for which they realize the necessary data and process translation. A successful architectural paradigm for the interoperability system is the *(Enterprise) Service Bus*. A service bus introduces a reference communication protocol and data model to translate to and from, as well as a set of commodity services such as service repository, enforcing quality of service and service composition. Conceptually, the advantage of the service bus that is well illustrated by the analogy with the hardware bus from which it derives, is that it acts as a pivot communication protocol to which networked systems may plug into. Then, still from a conceptual perspective, a networked system becomes interoperable "simply" by implementing a binding component that translates the system's protocol to that of the bus. It is important to highlight that the service bus is a solution to middleware-protocol interoperability; it does not deal with application-layer interoperability [3], although nothing prevents the introduction of higher-level domain-specific buses.

This paper is specifically about that topic: introducing a "*social communication bus*" to allow interoperability across computer-mediated social communication paradigms. Our work is motivated by our research effort within the AppCivist project (http://www.appcivist.org/) [11]. AppCivist provides a software platform for participatory democracy that leverages the reach of the Internet and the powers of computation to enhance the experience and efficacy of civic participation. Its first instance, AppCivist-PB, targets participatory budgeting, an exemplary process of participatory democracy that lets citizens prepare and select projects to be implemented with public funds by their cities [12]. For city-wide engagement, AppCivist-PB must enable citizens to participate with the Internet-based communication services they are the most comfortable with. In current practice, for example, seniors and teenagers (or youngsters under 18) are often the most common participants of this process [13], and their uses of technology can be fairly different. While seniors prefer traditional means of communication like phone calls and emails [2], a typical teenager will send and receive 30 texts per day [14]. The need for interoperability in this context is paramount since the idea is to include people in the participatory processes without leaving anyone behind. This has led us to revisit the service bus paradigm, for the sake of social communication across communities, to gather together the many communities of our cities.

The contributions of our paper are as follows:

- *Social communication paradigms*: Section II surveys the various forms of computer-mediated social communication supported by today's software services and tools. We then make an analogy with the communication paradigms implemented by middleware technologies, thereby highlighting that approaches to middleware interoperability conveniently apply to computer-mediated social communication interoperability.
- *Social Communication Bus architecture*: Section III then revisits the service bus paradigm for the domain-specific context of computer-mediated social interactions. We specifically build on the XSB bus [15], [16] that supports interoperability across interaction paradigms as opposed to interoperability across heterogeneous middleware protocols implementing the same paradigm. The proposed bus architecture features the traditional concepts of bus protocols and binding components, but those are customized for the sake of social interactions whose couplings differ along the social and presence dimensions.
- *Social Communication Bus instance for participatory democracy*: Section IV refines our bus architecture, introducing the Social-MQ implementation that leverages state of the art technologies. Section V then introduces how Social-MQ is used by the AppCivist-PB platform to enable reaching out a larger community of citizens in participatory budgeting campaigns.

Finally, Section VI summarizes our contributions and introduces some perspectives for future work.

II. Computer-mediated Social Communication

A. Computer-mediated Social Communication: An Overview

Social communication technologies change the way humans interact with each other by influencing identities, relationships, and communities [17]. Any human communication achieved through, or with the help of, computer technology is called *computer-mediated communication* [17], or as we call it in our work, *computer-mediated social communication* to highlight the fact that we are dealing with human communication. In this paper, we more specifically focus on text- and voice-based social communication technologies. These social communication technologies are usually conceived as Internet-based services - which we call *communication services* - that allow individuals to communicate between them [18]. Popular communication services include: Skype, which focuses on video chat and voice call services; Facebook Messenger, Telegram, WhatsApp, Slack, and Google Hangouts, which focus on instant messaging services; Twitter, which enables users to send and read short (140-character) messages; email, which lets users exchange messages, and SMS, which provides text messaging services for mobile telephony and also for the Web.

Depending on the communication service, users can send messages directly to each other or to a group of users; for example, a user can send an email directly to another user or to a mailing list where several users participate. In the former case, the users communicating via direct messaging "know each other". It does not mean that they have to know each other personally, it means they have to know the address indicating where and how to send messages directly to each other. In the latter example, communication is achieved via an intermediary: the mailing list address. In this case, senders

do not specify explicitly the receivers of their messages; instead, they only have to know the address of the intermediary to which they can send messages. The intermediary then sends messages to the relevant receivers, or receivers ask the intermediary for messages they are interested in. Another example of an intermediary is Twitter, where users can send messages to a topic. Interested users can subscribe to that topic and retrieve published messages.

Overall, existing communication services may be classified according to the types of interactions they support [19]: *interpersonal non-mediated communication*, where individuals interact directly; *impersonal group communication*, where people interact within a group; and *impersonal notifications*, where people interact in relation with some events to be notified. Our goal is then to leverage the technical interoperability solutions introduced for distributed systems for the specific domain of computer-mediated social communication so as to enable users to interact across communication services.

B. Computer-mediated Social Communication: A Technical Perspective

Communication protocols underlying distributed systems may be classified along the following coupling dimensions [15], [16], [20]:

- *Space coupling*: A tight (resp. loose) space coupling means that the sender and target receiver(s) need (resp. do not need) to know about each other to communicate.
- *Time coupling*: A tight time coupling indicates that the sender and target receiver(s) have to be online at same time to communicate, whereas a loose space coupling allows the receiver to be offline at the time of the emission; the receiver will receive messages when it is online again.
- *Synchronization coupling*: Under a tight synchronization coupling, the sender is blocked while sending a message and the receiver(s) is(are) blocked while waiting for a message. Under a loose synchronization coupling, the sender is not blocked, and the target receiver(s) can get messages asynchronously while performing some concurrent activity.

Following the above, we may define the coupling dimensions associated with computer-mediated social communication as:

- *Social coupling*: It is analogous to space coupling and refers to whether or not participants need to know each other to communicate.
- *Presence coupling*: It is analogous to the time coupling concept and refers to whether participants need to interact simultaneously.
- *Synchronization coupling*: Since we are addressing human interacting components, the synchronization coupling is alway loose since humans can do other activities after sending a message or while waiting for one. Hence, we do not consider this specific coupling in the remainder.

We may then characterize the types of interactions of communication services in terms of the above coupling (see Table I for a summary and Table II for the related classification of popular services):

	Social	*Presence*
Interpersonal non-mediated communication	tight	loose/tight
Impersonal group communication	loose	loose/tight
Impersonal notifications	loose	loose

TABLE I: Properties of computer-mediated social interactions

- *Interpersonal non-mediated communication:* Communicating parties need to know each other. Thus, the social coupling is tight. However, the presence coupling may be either tight or loose. Communication services enforcing a tight presence coupling relate to Video/voice calls and chat systems. On the other hand, base services like email, SMS, and instant messaging adopt a loose presence coupling.
- *Impersonal group communication:* The social coupling is loose because any participant may communicate with a group without the need of knowing its members. A space serves as an area that holds all the information making up the communication. To participate, users modify the information in the space. The presence coupling may be either loose or tight. As an example of tight presence coupling, shared meeting notes may be deleted once a meeting is over, so that newcomers cannot read it. Similarly, newcomers in a Q&A session cannot hear previous discussions. In a different situation, a service may implement loose presence coupling so that a participant (group member) can write a post-it note and let it available to anybody entering the meeting room. In addition, groups can be either closed or open [21]. In a closed group, only members can send messages. In an open group, non-members may also send messages to the group. Video/voice conferences and real-time multi-user chat systems are examples of group communication with a tight presence coupling. Message forums, file sharing, and multi-user messaging systems are examples of group communication with a loose presence coupling.
- *Impersonal notifications:* The social and presence coupling are loose. Participants do not need to know each other to interact. They communicate on the basis of shared interests (*aka* hashtags or topics). Twitter and Instagram are popular examples of such services.

C. Communication Service Interoperability

In general, users prefer a type of social interaction over the others [2], [14], [22]. This preference translates in favoring certain communication services. For example, someone may want to never interact directly and thus uses email whenever possible. Further, the adoption of specific communication service instances for social interactions increasingly limits the population of users with which an individual can communicate. Our work focuses on the study of interoperability across communication services, including services promoting different types of social interaction. This is illustrated in

	Tight presence coupling	*Loose presence coupling*
Interpersonal non-mediated communication	Skype call, Phone call Google Hangouts	Email, SMS, WhatsApp Facebook Messenger Google Hangouts Telegram, Slack
Impersonal group communication	Skype Group call Google Hangouts	WhatsApp Group Chat Telegram Group, Slack Facebook Groups Google Hangouts
Impersonal notifications	-	Twitter, Instagram Facebook Timeline

TABLE II: Classification of popular communication services

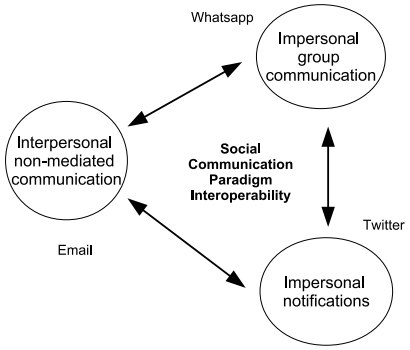

Fig. 1: Social communication interoperability

Figure 1 where users communicate although they employ different types of interactions via email, Twitter and Whatsapp.

We then need to study the extent to which different types of social interaction may be reconciled and when it is appropriate to synthesize the corresponding communication protocol adaptation. To do so, we build upon the *eXtensible Service Bus (XSB)* [15], [16] that is an approach to reconcile the middleware protocols run by networked systems across the various coupling dimensions (i.e., space, time, synchronization). This leads us to introduce the Social communication bus paradigm.

III. The Social Communication Bus

A. The eXtensible Service Bus

The *eXtensible Service Bus (XSB)* [15], [16] defines a connector that abstracts and unifies three interaction paradigms found in distributed computing systems: *client-server*, a common paradigm for Web services where a client communicates directly with a server; *publish-subscribe*, a paradigm for content broadcasting; and *tuple-space*, a paradigm for sharing data with multiple users who can read and modify that data. XSB is implemented as a common bus protocol that enables interoperability among services employing heterogeneous interactions following one of these computing paradigms. It also provides an API based on the `post` and `get` primitives to abstract the native primitives of the client-server (`send` and `receive`), publish-subscribe (`publish` and `retrieve`), and tuple-space interactions (`out`, `take`, and `read`).

In this work, we present the *Social Communication Bus* as a higher-level abstraction: XSB abstracts interactions of distributed computing interaction paradigms, while the Social Communication Bus abstracts interaction at the human level, that is, the computer-mediated social communication. Nonetheless, the Social Communication Bus relies on the XSB architectural paradigm. Most notably, the proposed Social Communication Bus inherits from XSB the approach to cross-paradigm interoperability that allows overcoming the coupling heterogeneity of protocols.

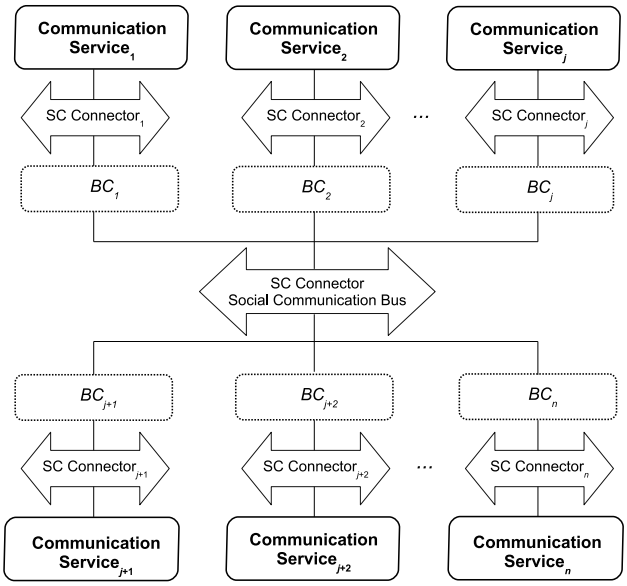

Fig. 2: The Social Communication Bus architecture

B. Social Communication Bus Architecture

Figure 2 introduces the architecture of the Social Communication Bus. The bus revisits the integration paradigm of the conventional Enterprise Service Bus [23] to enable interoperability across the computer-mediated social communication paradigms presented in Section II and concrete communication services implementing them.

In more detail and as depicted, the Social Communication Bus implements a common intermediate bus protocol that facilitates the interconnection of heterogeneous communication services: plugging-in a new communication service only requires to implement a conversion from the protocol of the service to that of the bus, thus considerably reducing the development effort. This conversion is realized by a dedicated component, called *Binding Component (BC)*, which connects the communication service to the Social Communication Bus. The binding that is implemented then overcomes communication heterogeneity at both abstract (i.e., it solves coupling mismatches) and concrete (i.e., it solves data and protocol message mismatches) levels. The BCs perform the bridging between a communication service and the Social Communication Bus by relying on the *SC connectors*. A SC connector provides access to the operations of a particular communication service and to the operations of the Social Communication Bus. Communi-

cation services can communicate in a loosely coupled fashion via the Social Communication Bus.

The Social Communication Bus architecture not only reduces the development effort but also allows solving the interoperability issues presented in Section II-C as follows:

- *Social coupling mediation*. The Social Communication Bus acting as an intermediary between communication services allows the social coupling mediation by decoupling senders and receivers. A communication service never sends any messages directly to another. Instead, a communication service can only send messages to an *address* in the Social Communication Bus. Then, the Social Communication Bus pushes these messages to one or more communication services, or communication services can retrieve them.
- *Presence coupling mediation*. Decoupling senders and receivers also allows the presence coupling mediation. Since the Social Communication Bus acts as an intermediary for handling messages, it can do the temporal mediation of those messages so that communication services with incompatible presence coupling can interact at different times.

C. The API for Social Communication (SC API)

To reconcile the different interfaces of communication services and connect them to the Social Communication Bus, we introduce a generic abstraction. The proposed abstraction comes in the form of a *Social Communication Application Programming Interface (SC API)*. The SC API abstracts communication operations executed by the human user of a communication service, such as, e.g., sending or receiving a message. We also assume that these operations are exported by the communication service in a public (concrete) API, native to the specific communication service. This enables deploying interoperability artifacts (i.e., BCs) between heterogeneous communication services that leverage these APIs.

The SC API expresses basic common end-to-end interaction semantics shared by the different communication services, while it abstracts potentially heterogeneous semantics that is proper to each service. The SC API relies on the two following basic primitives:

- a `post()` primitive employed by a communication service to send a message;
- a `get()` primitive employed by a communication service to receive a message.

To describe a communication service according to the SC API, we propose a generic interface description (SC-IDL). This interface describes a communication service's operations, including the name and type of their parameters. The description is complemented with the following communication service information: `name`, its name; `address`, the address of the endpoint of its public API; `protocol`, its middleware protocol (e.g., HTTP, SMTP, AMQP, MQTT); and `social_properties`, which specifies if the communication service handles messages when its users are offline.

D. Higher-order Binding Components

BCs are in charge of the underlying middleware protocol conversion and application data adaptation between a communication service and the Social Communication Bus. As presented in XSB, BCs do not alter the behavior and properties of the communication services associated with them, they do not change the end-to-end communication semantics; however, since these communication services can be heterogeneous and can belong to different providers, it may be desirable to improve their end-to-end semantics to satisfy additional user requirements and to mediate social and presence coupling incompatibility. To this end, we introduce the *higher-order BCs*, which are BCs capable of altering the perceived behavior of communication services. We propose the two following higher-order BCs capabilities:

- *Handling offline receiver*: this case is related to the mediation of presence coupling in computer-based social communication, and it occurs when the receiver is not online and he is using a communication service that does not support offline message reception. Even though the server hosting this communication service is up and running, it discards received messages if the recipient is offline. A higher-order BC will send undelivered messages when the receiver logs back into the system. We do not enforce this capability; instead, we let users decide if they want to accept offline messages or not.
- *Handling unavailable receiver*: this case is similar to the previous one but from a computing perspective, related to fault tolerance; for example, the server providing the receiver service is down, or there is no connectivity between the BC and the receiver. The BC will send undelivered messages once the receiver service is available again. In contrast to the previous case, this capability is provided by higher-order BCs by default.

IV. Implementation

A. Social-MQ: An AMQP-based Implementation of the Social Communication Bus

Social-MQ leverages the AMQP protocol as the Social Communication Bus. AMQP has several open source implementations, and its direct and publish/subscribe models serve well the purpose of social interactions: interpersonal non-mediated communication, impersonal group communication, and impersonal notifications. Additionally, AMQP has proven to have good reliability and performance in real-world critical applications in a variety of domains [24]. We use RabbitMQ [25] as the AMQP implementation.

The bus comes along with a BC generator (see Figure 3). The generator takes as input the description of a communication service (SC-IDL), chooses the corresponding SC connector from the Implementation Pool, and produces a Concrete BC connecting the communication service with Social-MQ. The BC generator is implemented on the Node.js [26] platform, which is based on the Chrome JavaScript virtual machine. Node.js implements the reactor pattern that allows

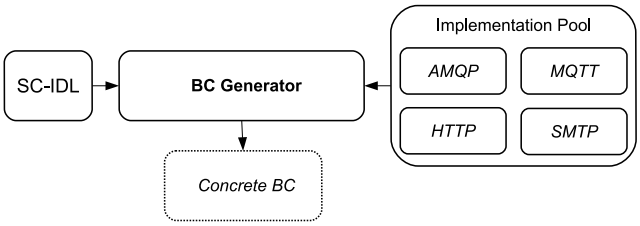

Fig. 3: BC Generator

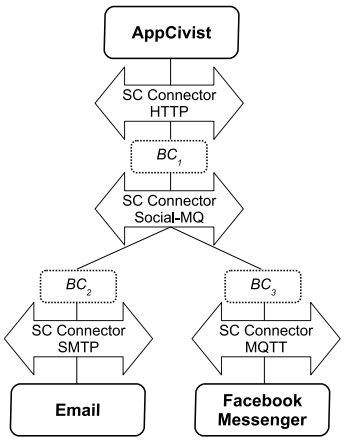

Fig. 4: Social-MQ architecture

building highly concurrent applications. Currently, BCs are generated for the Node.js platform uniquely. We intend to support other languages or platforms in future versions of the bus. Social-MQ currently supports four middleware protocols: AMQP, HTTP, MQTT, and SMTP.

Figure 4 illustrates the connection of communication services to Social-MQ. All the associated BCs are AMQP publishers and/or subscribers so that they can communicate with Social-MQ. In more detail:

- BC_1 exposes an HTTP endpoint so that the HTTP communication service can send messages to it, and it can act as HTTP client to post messages to the communication service.
- BC_2 acts as an SMTP server and client to communicate with Email;
- BC_3 has MQTT publisher and subscriber capabilities to communicate with the MQTT communication service.

The above BCs are further refined according to the actual application data used by AppCivist, Email, and Facebook Messenger.

The interested reader may find a set of BCs generated by Social-MQ at https://github.com/rafaelangarita/bc-examples. These BCs can be executed and tested easily by following the provided instructions.

B. Social-MQ Implementation of Social Interaction Mediation

The loosely coupled interaction model between communication services provided by Social-MQ allows the mediation between the various types of social interactions supported by

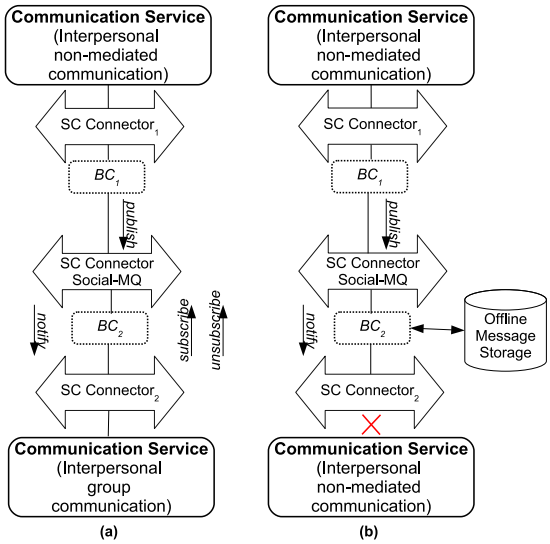

Fig. 5: (a) Space coupling mediation; (b) Presence coupling mediation

popular communication services. More specifically, Social-MQ handles social and presence coupling incompatibilities as follows:

- *Social coupling mediation:* In the publish/subscribe model implemented by Social-MQ (Figure 5 (a)), senders publish a message to an *address* in Social-MQ, instead of sending it directly to a receiver. Receivers can subscribe to this address and be notified when new messages are published. This way, all social communication paradigms can interact using the publish/subscribe model.
- *Presence coupling mediation*. When a communication service cannot receive messages because it is not available or its user is offline, messages intended for it are sent to a database to be queried and sent when the communication service can receive messages again (Figure 5 (b)).

V. The AppCivist Use Case

A. The AppCivist Platform for Participatory Democracy

To illustrate our approach, we elaborate on the use of Social-MQ to enable the AppCivist application for participatory democracy [11], [12] to interoperate with various communication services. This way, the citizens participating to AppCivist actions may keep interacting using the social media they prefer.

AppCivist allows activist users to compose their own applications, called *Assemblies*, using relevant Web-based components enabling democratic assembly and collective action. AppCivist provides a modular platform of services that range from proposal making and voting to communication and notifications. Some of these modules are offered as services implemented within the platform itself (e.g., proposal making), but for others, it relies on existing services. One of such cases is that of communication and notifications. Participatory processes often rely on a multitude of diverse users, who not

always coincide in their technological choices. For instance, participatory budgeting processes involve people from diverse backgrounds and of all ages: from adolescents (or youngsters under 18), to seniors [13]. Naturally, their technology adoption can be fairly different. While seniors favor traditional means of communication like phone calls and emails [2], a typical teenager will send and receive 30 texts per day [14]. The need for interoperability in this context is outstanding, and the Social Communication Bus is a perfect fit, with its ability to bridge communication services that power computer-based social communication. In the following, we discuss three communication scenarios: (i), impersonal notifications interconnected with impersonal group communication; (ii), interpersonal non-mediated communication interconnected with impersonal group communication; and (iii), interpersonal non-mediated communication interconnected with impersonal notifications. The last scenario also illustrates the presence coupling mediation feature of Social-MQ.

B. Impersonal Notifications Interconnected with Impersonal Group Communication

In this scenario, users of AppCivist interact via the impersonal notification paradigm by using a notification system implemented using AMQP as described in Listing 1. This notification system sends messages to concerned or interested users when different events occur in AppCivist; for example, when a user posts a new forum message. This scenario is illustrated in Figure 6 (a). AppCivist is connected to Social-MQ via BC_1; however, there is no need of protocol mediation, since both AppCivist and Social-MQ use AMQP. Mailing List is another system which exists independently of AppCivist. It is a traditional mailing list in which users communicate with each other using the impersonal group communication paradigm by sending emails to the group email address. Mailing List is connected to Social-MQ via BC_2, and it is described in Listing 2. It accepts receiving messages whether or not receivers are online or offline (`properties.offline.handling = true`, Listing 2). It is due to the loose presence coupling nature of email communication.

Now, suppose users in Mailing List want to be notified when a user posts a new forum message in AppCivist. Then, since AppCivist is an AMQP-based notification system, BC_1 can act as AMQP subscriber, receive notifications of new forum posts, and publish them in Social-MQ. In the same way, BC_2 acts as AMQP subscriber and receives notifications of new forum posts; however, this time BC_2 receives the notifications from Social-MQ. Finally, BC_2 sends an email to Mailing List using the SC Connector SMTP.

Listing 1: AppCivist AMQP SC-IDL

```
{
 "name":"AppCivist"
 "address":"appcivist.littlemacondo.com:5672",
 "protocol":"AMQP",
 "operations":[
   "notify":{
     "interaction":"one−way",
     "type":"data",
     "scope":"assembly_id.forum.post",
     "post_message":[
     {"name":"notification",
      "type":"text",
      "optional":"false"}]
   }
 ],
 "properties":[
   {"offline":"true"}
  ]
}
```

Listing 2: Mailing List SC-IDL

```
{
 "name":"Mailing List",
 "address":"mailinglist_server",
 "protocol":"SMTP",
 "operations":[
   "receive_email":{
     "interaction":"one−way",
     "type":"data",
     "scope":"mailinglist_address",
     "get_message":[
       {"name":"subject",
        "type":"emailSubject",
        "optional":"true"},
       {"name":"message",
        "type":"messageBody",
        "optional":"true"},
       {"name":"attachment",
        "type":"file",
        "optional":"true"}]
   },
   "send_email":{
     //same as receive_email
   }
 ],
"properties":[
   {"offline":"true"}
  ]
}
```

C. Interpersonal Non-mediated Communication Interconnected with Impersonal Group Communication

In the scenario illustrated in Figure 6 (b), there is an AppCivist communication service called Weekly Notifier. It queries the AppCivist database once a week, extracts the messages posted in AppCivist forums during the last week, builds a message with them, and sends the message to concerned users using interpersonal non-mediated communication via HTTP. That is, it is an HTTP client, so it sends the message to an HTTP server.

Now, suppose we want Weekly Notifier to communicate with Mailing List. BC_1 exposes an HTTP endpoint to which Weekly Notifier can post HTTP messages. Differently from the previous case, we need to modify the original Weekly Notifier communication service since it needs to send messages to the endpoint exposed by BC_1 and it needs to specify Mailing List as a recipient.

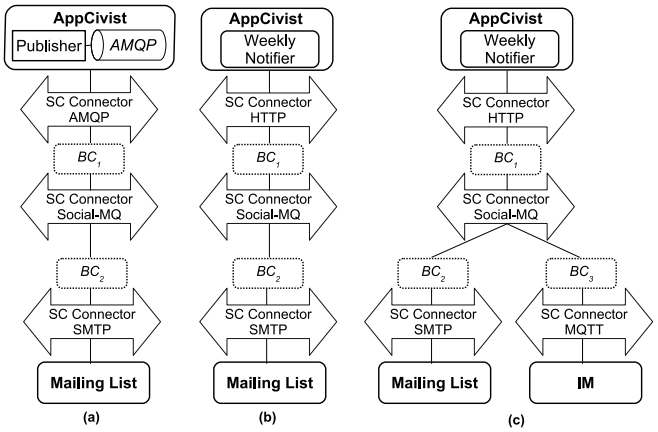

Fig. 6: Use Cases: (a), impersonal notifications interconnected with impersonal group communication; (b), interpersonal non-mediated communication interconnected with impersonal group communication; and (c), interpersonal non-mediated communication interconnected with impersonal notifications

Listing 3: AppCivist HTTP SC-IDL

```
{
  "name":"AppCivist"
  "address":"",
  "protocol":"HTTP",
  "operations":[
    "notify":{
      "interaction":"one−way",
      "type":"data",
      "post_message":[
      {"name":"notification",
       "type":"text",
       "optional":"false"}]
    }
  ],
  "properties":[
    {"offline":"false"}
   ]
}
```

D. Interpersonal Non-mediated Communication Interconnected with Impersonal Notifications

After having introduced the previous scenario, we can pose the following question: what if messages sent by Weekly Notifier must be sent to multiple receivers? Should Weekly Notifier know them all and send the message individually to each one of them? Independently of the communication services registered in Social-MQ and their social communication paradigms, they can all interact in a fully decoupled fashion in terms of social coupling.

Social-MQ takes advantage of the *exchanges* concept of AMQP, which are entities where messages can be sent. Then, they can route messages to receivers, or interested receivers can subscribe to them. In the scenario illustrated in Figure 6 (c), Weekly Notifier sends HTTP messages directed to the Social-MQ exchange named *AppCivist weekly notification*. Interested receivers can then subscribe to `AppCivist weekly notification` to receive messages from Weekly Notifier. Finally, Mailing List and the instant messaging communication service, IM (Listing 4), can subscribe to `AppCivist weekly notification` via their corresponding BCs.

Listing 4: IM SC-IDL

```
{
  "name":"IM",
  "address":"mqtt.example",
  "protocol":"MQTT",
  "operations":[
    "receive_message":{
      "interaction":"one−way",
      "type":"data",
      "scope":"receiver_id",
      "get_message":[
      {"name":"message",
       "type":"text",
       "optional":"false"}]
    },
    "receive_attachement":{
      "interaction":"one−way",
      "type":"data",
      "scope":"receiver_id",
      "get_message":[
      {"name":"message",
       "type":"file",
       "optional":"false"}]
    },
    "send_message":{
      //same as receive_message
    },
    "send_attachement":{
      //same as receive_attachement
    }
  ],
  "properties":[
    {"offline":"false"}
   ]
}
```

E. Assessment

In this section, we have studied three case studies illustrating how Social-MQ can solve the problem of computer-mediated social communication interoperability. These case studies are implemented for the AppCivist application for participatory democracy. As a conclusion, we argue that: (i), Social-MQ can be easily integrated into existing or new systems since it is non-intrusive and most of its processes are automated; (ii), regarding performance and scalability, Social-MQ is implemented on top of technologies that have proven to have high performance and scalability in real-world critical applications; and (iii), Social-MQ allows AppCivist users to continue using the communication service they prefer, enabling to reach a larger community of citizens, and promoting citizen participation.

VI. Conclusion and Future Work

We have presented an approach to enable social communication interoperability in heterogeneous environments. Our

main objective is to let users use their favorite communication service. More specifically, the main contributions of this paper are: a classification of the social communication paradigms in the context of computing; an Enterprise Service Bus-based architecture to deal with the social communication interoperability; and a concrete implementation of the Social Communication Bus studying real-world scenarios in the context of participatory democracy.

For our future work, we plan to present the formalization of our approach and to incorporate popular communication services such as Facebook Messenger, Twitter, and Slack. The interoperability with these kinds of services poses additional challenges, since the systems they belong to can be closed; for example, Facebook Messenger allows sending and receiving messages only to and from participants that are already registered in the Facebook platform. Another key issue to study is the security & privacy aspect of the Social Communication Bus to ensure that privacy needs of users communicating across heterogeneous social media are met. Last but not least, our studies will report the real-world experiences of AppCivist users regarding the Social Communication Bus.

ACKNOWLEDGMENTS

This work is partially supported by the Inria Project Lab CityLab (citylab.inria.fr), the Inria@SiliconValley program (project.inria.fr/siliconvalley) and the Social Apps Lab (citris-uc.org/initiatives/social-apps-lab) at CITRIS at UC Berkeley. The authors also acknowledge the support of the CivicBudget activity of EIT Digital (www.eitdigital.eu).

REFERENCES

[1] J.-C. Pillet and K. D. A. Carillo, "Email-free collaboration: An exploratory study on the formation of new work habits among knowledge workers," *International Journal of Information Management*, vol. 36, no. 1, pp. 113 – 125, 2016.

[2] A. Dickinson and R. L. Hill, "Keeping in touch: Talking to older people about computers and communication," *Educational Gerontology*, vol. 33, no. 8, pp. 613–630, 2007.

[3] V. Issarny, A. Bennaceur, and Y.-D. Bromberg, "Middleware-layer connector synthesis: Beyond state of the art in middleware interoperability," in *SFM-11: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems âĂŞ Connectors for Eternal Networked Software Systems*. Springer Berlin Heidelberg, 2011, vol. 6659, pp. 217–255.

[4] B. Spitznagel and D. Garlan, "A compositional formalization of connector wrappers," in *Proceedings of the 25th International Conference on Software Engineering*, ser. ICSE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 374–384.

[5] R. Mateescu, P. Poizat, and G. Salaün, "Adaptation of service protocols using process algebra and on-the-fly reduction techniques," *IEEE Transactions on Software Engineering*, vol. 38, no. 4, pp. 755–777, July 2012.

[6] C. Gierds, A. J. Mooij, and K. Wolf, "Reducing adapter synthesis to controller synthesis," *IEEE Trans. Serv. Comput.*, vol. 5, no. 1, pp. 72–85, Jan. 2012.

[7] A. Bennaceur and V. Issarny, "Automated synthesis of mediators to support component interoperability," *IEEE Transactions on Software Engineering*, vol. 41, no. 3, pp. 221–240, 2015.

[8] A. Bennaceur, E. Andriescu, R. S. Cardoso, and V. Issarny, "A unifying perspective on protocol mediation: interoperability in the future internet," *Journal of Internet Services and Applications*, vol. 6, no. 1, p. 12, 2015. [Online]. Available: http://dx.doi.org/10.1186/s13174-015-0027-3

[9] G. Wiederhold, "Mediators in the architecture of future information systems," *Computer*, vol. 25, no. 3, pp. 38–49, Mar. 1992.

[10] G. Bouloukakis, N. Georgantas, S. Dutta, and V. Issarny, "Integration of Heterogeneous Services and Things into Choreographies," in *14th International Conference on Service Oriented Computing (ICSOC)*, Banff, Canada, Oct. 2016.

[11] A. Pathak, V. Issarny, and J. Holston, "AppCivist - A Service-oriented Software Platform for Socially Sustainable Activism," in *International Conference on Software Engineering (ICSE), Software Engineering in Society (SEIS) Track*, Florence, Italy, May 2015. [Online]. Available: https://hal.inria.fr/hal-01109314

[12] J. Holston, V. Issarny, and C. Parra, "Engineering software assemblies for participatory democracy: The participatory budgeting use case," in *International Conference on Software Engineering (ICSE), Software Engineering in Society (SEIS) Track*. Austin, Texas: ACM, 2016, pp. 573–582.

[13] C. Hagelskamp, C. Rinehart, R. Silliman, and D. Schleifer, "Public Spending, by the People. Participatory Budgeting in the United States and Canada in 2014 - 15," Public Agenda, Tech. Rep., 2016.

[14] A. Lenhart, "Teens, social media & technology overview 2015," Pew Research Center. http://www.pewinternet.org/2015/04/09/teens-social-media-technology-2015/, 2015, Accessed: 2016-09-30.

[15] N. Georgantas, G. Bouloukakis, S. Beauche, and V. Issarny, "Service-oriented distributed applications in the future internet: The case for interaction paradigm interoperability," in *Service-Oriented and Cloud Computing: Second European Conference*. Málaga, Spain: Springer Berlin Heidelberg, 2013, pp. 134–148. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40651-5_11

[16] A. Kattepur, N. Georgantas, G. Bouloukakis, and V. Issarny, "Analysis of timing constraints in heterogeneous middleware interactions," in *Service-Oriented Computing: 13th International Conference, ICSOC 2015*. Goa, India: Springer Berlin Heidelberg, 2015, pp. 36–52.

[17] C. Thurlow, L. Lengel, and A. Tomic, *Computer mediated communication: Social interaction and the internet*. London: Sage, 2004.

[18] A. Richter and M. Koch, "Functions of social networking services," in *Proc. Intl. Conf. on the Design of Cooperative Systems*, 2008, pp. 87–98.

[19] J. B. Walther, "Computer-mediated communication: Impersonal, interpersonal, and hyperpersonal interaction," *Communication Research*, vol. 23, no. 1, pp. 3–43, 1996. [Online]. Available: https://www.learntechlib.org/p/80853

[20] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003.

[21] L. Liang, S. T. Chanson, and G. W. Neufeld, "Process groups and group communications: Classifications and requirements," *Computer*, vol. 23, no. 2, pp. 56–66, Feb. 1990.

[22] A. N. Joinson, "Self-esteem, interpersonal risk, and preference for e-mail to face-to-face communication," *CyberPsychology & Behavior*, vol. 7, no. 4, pp. 472–478, 2004.

[23] D. Chappell, *Enterprise service bus*. O'Reilly Media, Inc., 2004.

[24] S. Appel, K. Sachs, and A. Buchmann, "Towards Benchmarking of AMQP," in *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '10. New York, NY, USA: ACM, 2010, pp. 99–100.

[25] "Rabbitmq," https://www.rabbitmq.com, Accessed: 2016-09-30.

[26] "Node.js," https://nodejs.org, Accessed: 2016-09-30.