A Model-Driven Approach for Real-time Role-Based Communication

Marcelo B. Azevedo Vieira[1] **, Sérgio T. Carvalho**[1] **,**
Fábio M. Costa[1] **, David Bromberg**[2]

[1]Instituto de Informática, Universidade Federal de Goiás
Campus Samambaia, Goiânia-GO, 74690-900 – Brazil

[2]IRISA, University of Rennes 1, Rennes, France

marcelobazevedo@gmail.com, {sergio|fmc}@inf.ufg.br, david.bromberg@irisa.fr

Abstract. *Recent years have seen the inception of many domain-specific modelling languages, enabling to overcome some of the main difficulties found in software development. The use of models has a particular impact on the implementation phase, as models tend to be closer to the problems to be solved than code. This paves the way to enable application construction by non-experts in software development, such as domain specialists. In this paper, we exploit the use of models in the domain of real-time communication, which poses significant challenges for application construction due to the multitude and intricacy of the technologies involved. We propose RBCML, a communication modelling language for the high-level specification of real-time communication sessions based on the roles that users play in the sessions. The language is processed using a combination of partial code generation and dynamic model interpretation, resulting in the construction of fully functional communication applications. The paper describes RBCML and its implementation on top of W3C's Web Real-Time Communication protocols (WebRTC). An evaluation is presented to compare the use of RBCML with code-based development and to characterize the performance of communication session establishment using the language.*
Keywords: *communication models, multimedia services, real-time communication.*

1. Introduction

Recent years have seen the development and popularisation of several applications for real-time communication (RTC), enabling person-to-person interaction using messaging, audio/video streams and document exchange. Among them, we can highlight MSN Messenger, which dominated the market during the last decade, and Skype, WhatsApp, Facebook Messenger, Hangouts Meet, Slack, Zoom Meetings, and Cisco Webex, to name just a few, which dominate the market nowadays in different segments. More recently, the increased demand for this kind of applications has become evident due to the need to maintain person-to-person interaction, both formal and informal, during the challenging times of the 2020 coronavirus pandemic. Advances in media encoding and compression, together with new communication protocols and significant improvements on Internet connectivity and bandwidth, have enabled such applications to provide increased quality.

One notable advance was the adoption of peer-to-peer architectures, which resulted in lower latency and better user experience.

With the advent of W3C's WebRTC[1], existing real-time communication applications, including some of those mentioned above, evolved to allow multi-party communication using standard Web browsers, without the need to install specific tools or plug-ins. WebRTC APIs and protocols provide a standard way to add the ability to establish custom browser-to-browser communication sessions (consisting of audio and video streams, along with the exchange of text messages and files) as part of any Web-based application. Please note that we use the term "real-time communication" in this paper for alignment with WebRTC literature. We nevertheless acknowledge that a more precise term was proposed in the ITU-T F.703 Recommendation: "Multimedia Conversational Services" [ITU-T 2000].

The demand for embedded conversational features has also been partially met by existing systems, such as Skype, with APIs and services that enable their use from inside other applications. However, this kind of integration may not be flexible enough, as it has to adhere to predefined user interaction flows and authentication schemes, thus hindering customisation. The ability to flexibly compose real-time communication primitives is thus an essential requirement to enable the creation of conversational features tailored to each application domain and usage scenario. We must, however, overcome the complexities of building such customised applications, enabling both novice developers and sophisticated end-users to build communication applications for specific purposes.

In this paper, we propose a novel approach to build customised real-time communication applications based on the use of models and user roles. We propose a DSML (domain-specific modelling language) to eliminate the accidental complexities of building such applications using general-purpose programming languages. The DSML, called RBCML, provides a cohesive set of building blocks for the flexible configuration of real-time communication sessions, enabling the distinction of the different roles that users play in session control and communication. It enables modelling of a variety of session configurations to suit different application domains and scenarios. Some examples include virtual classrooms (with users taking up the roles of teacher and *students*), remote PhD defences (where applicable roles are *candidate, advisor, examiner* and *member of the audience*), and telemedicine (where the roles might be *assisting doctor, specialist doctor, patient*, and *nurse*). Each scenario may be described by a separate model that specifies how each role participates in the session, including the connections among them and the number of participants that each role admits.

Execution of RBCML models uses a combination of code generation, which handles the overall structure of a communication session, and dynamic model interpretation, which parameterises and instantiates the different elements of the structure as the communication session at session establishment time, i.e., when peers join the session.

RBCML is independent of any particular communication platform. However, to demonstrate and validate it, we have built a prototype on top of the WebRTC platform, which is currently embedded in most mainstream Web browsers. As such, RBCML models are used to generate JavaScript code for the establishment of the data and peer con-

[1] https://webrtc.org

nections that make up WebRTC communication sessions. We demonstrate and evaluate RBCML using a common real-time communication scenario. The evaluation is twofold. Firstly, we present the results of an experiment carried out with a group of students, who developed an application for the given scenario using both our model-driven approach and the more conventional programming-based approach with JavaScript. We observed the performance of the students in both tasks and applied a questionnaire at the end. This allowed us to quantify the effectiveness of our approach. Secondly, we present the results of a quantitative evaluation of the performance of communication session establishment using our approach.

The remaining of this paper is organized as follows. Section 2 discusses the basic concepts and foundations for the work, while Section 3 discusses related work. Section 4 presents the design of RBCML, including its metamodel and concrete syntaxes, while Section 5 discusses its implementation. Finally, Section 6 presents the qualitative and quantitative evaluation of RBCML, based on its implementation on top of WebRTC, and Section 7 presents concluding remarks and discusses future work.

2. Background

2.1. Domain-Specific modelling Languages

Model-Driven Engineering (MDE) refers to software development approaches in which models are used as the primary development artefacts [Bézivin 2005], as opposed to being used just for documentation purposes. In most approaches, software models are gradually and automatically transformed into source code via a series of model-to-model and model-to-code transformations.

Often, models used in MDE focus on specific domains and are defined using DSMLs. As opposed to general-purpose modelling languages, such as UML, a DSML uses constructs that are specific to its application domain. Domain specificity helps in the association of precise semantics to the modelling constructs of the language, thus making it feasible to achieve full code generation or model interpretation. According to [de Farias et al. 2007], DSMLs must are through the use of metamodels that appropriately represent the concepts of the application domain. The use of a metamodel guarantees not only strong semantics for the domain-specific constructs, but also provides a precise abstract syntax for their representation.

DSMLs have been proposed for a variety of application domains, such as financial services, entertainment, smart spaces, and communication [Medvidovic and Taylor 1997]. Their power to raise the level of abstraction beyond coding, together with the ability to focus on a small number of constructs that are necessary for the domain at hand, is what makes them good candidates to enable user-centred application development, as proposed in this paper.

2.2. Peer-to-peer communication

The amount of data consumed by Internet users has grown steadily in the last decades, and trends indicate that this growth is bound to continue in the foreseeable future. Moreover, the number of users simultaneously consuming Internet content is continuously growing. Nevertheless, the number of servers that host the content does not grow at the same pace, negatively affecting service performance [Cho et al. 2006].

The concept of peer-to-peer (P2P) communication was proposed to alleviate this problem. Using P2P connections, users may transfer audio, video and data directly from one machine to another, without going through a server. This considerably reduces latency and, as a result, a substantial fraction of the data exchanges on the Internet take place in this fashion. P2P communication schemes have the following essential characteristics [Jia et al. 2017]: communicating endpoints (peers) are always directly connected (at the level of an overlay network); peers are responsible for their own data; peers may join or leave the network at any time; peers can act both as clients and servers; and there is autonomy regarding control and structure of the network, thus avoiding the need for central authorities.

2.3. Web Real-Time Communication

Web Real-Time Communication (WebRTC) is a standard Web technology that allows applications to make browser-to-browser calls using audio, video and generic data via peer-to-peer connections. WebRTC enables this feature without the need to use third-party applications, plug-ins or intermediate services [Sredojev et al. 2015][2]. The standard has three fundamental components, which support the different facets of real-time communication, and which can be accessed via a JavaScript API and embedded in HTML5 pages:

- **MediaStream** – allows a browser to access the media streams that originate from a device's camera and microphone;
- **RTCPeerConnection** – allows direct browser-to-browser connections across devices for the streaming of audio, video and data;
- **RTCDataChannel** – allows Web browsers to send and receive data files and text strings across peer-to-peer connections.

As opposed to pure peer-to-peer applications, in which nodes establish connections by directly contacting their peers, WebRTC requires the use of a separate signalling channel (and server) to negotiate connection parameters. This provides a level of security by avoiding the need for browsers to keep open ports for signalling. It also allows the use of straightforward routing strategies that enable connection among peers that lie behind NATs or firewalls [Tindall and Harwood 2015]. After negotiating the session parameters using an external signalling server, all further communication in the session occurs via peer-to-peer connections [Sredojev et al. 2015].

3. Related Work

[Clarke et al. 2006] propose a new paradigm for describing user-to-user communication, called *user-centric communication*. The centre point of this paradigm is a declarative DSML called Communication Modeling Language (CML), which is directly executable by its associated model execution engine, called Communication Virtual Machine (CVM). One of the main characteristics of CML is the abstraction of the devices and networks on which communication applications run, which facilitates the description of the communication logic of applications. CVM, in turn, facilitates user-centric communication by abstracting the underlying communication services and directly executing CML models in terms of the communication primitives they provide. The CML approach is more closely

[2]WebRTC is currently a W3C Candidate Recommendation and can be found at https://www.w3.org/TR/webrtc/

related to the approach proposed in this paper, so it is described more thoroughly in the remaining of this section.

CML models define both *communication schemas*, which describe the overall structure of a communication session, and *communication instances*, which are runtime descriptions of actual communication sessions that take place at a given moment. An instance is related to a schema in the same way as an object is related to a class in object-oriented languages. A communication instance captures all information that describes a communication session at a given time, such as the participants' IDs, device capabilities, and the data transferred across the connections. A schema, in turn, provides a more general specification of the configurations that are allowed in any given communication session instantiated from it.

CML has well-defined semantics and abstract syntax, which are described in terms of its metamodel. As for concrete syntax, it provides two options: G-CML, a graphical syntax that uses different shapes to represent the constructs employed in the description of communication sessions; and X-CML, an equivalent XML-based syntax, which also serves as the internal representation of models inside the execution engine. G-CML models provided as input to CVM are first translated into X-CML before execution.

Although CML inspired the design of RBCML, the two approaches differ in two crucial ways. Firstly, while CML describes communication sessions in terms of the actual users that should take part in the sessions, in RBCML, a communication session is described in terms of the roles. These can be instantiated (or not) as many times as the number of users playing them on a session (subject to cardinality constraints defined in the model). Thus communication models in RBCML are more generic than in CML. Secondly, while the runtime interpreter provided by CVM supports the execution of CML models, RBCML models are meant mainly for code generation, as described in Section 5.

[Contreras et al. 2016] present U-DSL, a modelling language for the domain of ubiquitous systems. The metamodel of the language has constructs to represent the different kinds of concepts that appear in ubiquitous computing environments, such as devices with their capabilities (e.g., sensors, actuators and controllers). It also defines constructs for monitoring and manipulating those concepts, such as location, configuration and connection. Similarly to RBCML, U-DSL has a concrete syntax that combines graphical and textual elements. The first-class constructs of the metamodel are represented in a graphical way using diagrams with boxes of different shapes and colours. In contrast, the characteristics of devices and services are represented using XML.

4. RBCML

RBCML is a novel DSML for the horizontal domain of real-time communication. It is intended for users who are not software developers but are specialists in their vertical application domains. Such users do not need to know the intricacies of real-time communication from a programming point of view. However, they have a solid knowledge of the requirements, general structure, and rules of communication in their respective domains of speciality. For instance, a specialist in the domain of law would know the communication requirements to set up a remote court trial involving users taking up the relevant roles (e.g., *judge*, *jury member*, *attorney*, *prosecutor*, and *defendant*). RBCML, therefore, enables domain specialists to model the general structure of communication sessions in

terms of user roles and the connections among them, together with the constraints that govern the interactions. A model can then be instantiated for any group of end-users that match, in a consistent way, the roles defined in the model.

4.1. The RBCML metamodel

The metamodel of RBCML is presented in Figure 1. It is purposely simple to limit the number of concepts that domain specialists need to master. Nevertheless, as shown in Section 6, it is powerful enough to enable a wide range of communication structures.

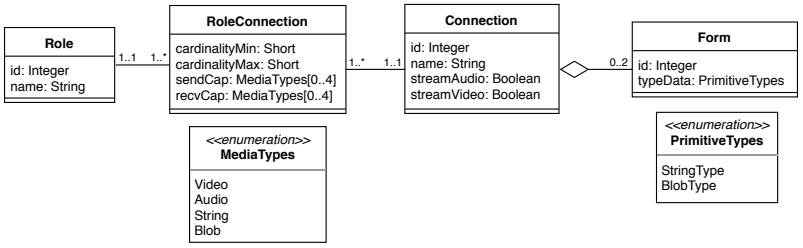

Figure 1. RBCML metamodel.

The central element of the metamodel is the Connection metaclass. A model for a communication session may have one or more instances of Connection, representing the actual communication between the users in a session. A connection may be used to transport *audio and video streams*, as well as data (character strings and files) embedded in *forms*, between the devices belonging to users that play specific roles in the session. These elements are specified by the `streamAudio` and `streamVideo` attributes of `Connection`, and by the metaclass Form, respectively. The user roles linked by a connection are specified using the meta-class Role, noting that a connection may link users that play one or more roles. The cardinality attribute of Role indicates how many users may play a given role in the same connection instance. Finally, it is important to note that the association between Role and Connection is realized through the auxiliary meta-class RoleConnection, so that a given role may have different cardinalities and media constraints in the different connections in which it participates.

Putting it all together, an RBCML model is what defines a communication session, which in turn may contain one or more connections to link the users of a group using a variety of configurations of media and data forms. In terms of logical structure, the group of users participating in a session may be further structured into subgroups by simply using a separate connection to link the users of each subgroup. The subgroups of a session may overlap, meaning that the same user can participate in more than one connection (playing the same or a different role). At runtime, a UI artefact (e.g., a set of buttons) generated from the model as part of the application allows the user to seamlessly switch communication from one connection to another within the same session.

The advantage of creating these multi-connection sessions using RBCML models is twofold. Firstly, end-users do not need to bother about managing separate application sessions to interact with different subsets of a group of users. A user only needs to accept an invitation to join a communication session (e.g., by pointing to the corresponding URL) and a complete, custom-built application is loaded into her browser, providing seamless access to all the related connections from a single place. In a typical scenario, there may

be a connection linking all members of the group, along with separate connections that enable private conversations between selected members. While such a scenario can be realized using several instances of a communication application, the approach supported by RBCML enables the establishment of the entire communication structure at once and coherently. Secondly, the approach is especially useful in the presence of specific communication structures and rules determined by regulations or common practice, such as in the examples given in Section 1. Such regulations can be captured in the form of a model that prescribes the acceptable structure of sessions (e.g., which user roles are enabled to send/receive media to/from which other roles). Furthermore, the generality of an RBCML model enables the use of the same model to automatically create different communication sessions, i.e., by instantiating the model for a different group of users, where different numbers of users fulfil each role, as determined by the specific application scenario (and allowed by the constraints expressed in the model).

4.2. RBCML Textual syntax: J-RBCML

While the metamodel defines the abstract syntax of the language, the concrete syntax is defined separately. We propose two alternatives in this respect: a graphical syntax, G-RBCML, meant for domain specialists that need to model communication sessions, and a textual syntax, which might also be used by domain specialists but is mainly used as an internal representation of models. The textual syntax, called J-RBCML, is based on the JavaScript Object Notation (JSON), thus using key-value pairs to represent the model elements. J-RBCML models are directly processed for code generation and interpretation. In the current implementation, models are provided as input in J-RBCML. In future work, as we complete the toolchain implementation, J-RBCML models will be generated from graphical models specified using G-RBCML. The syntax of G-RBCMLis described in [de Azevedo Vieira 2018].

Figure 2 presents an excerpt of a model of a communication session comprising two connections to support a hypothetical remote courtroom trial scenario based on video-conferencing. It specifies the major roles taking part in a typical courtroom trial, along with their media requirements. The Court Trial connection involves all the roles and supports both audio and video, as well as text and files (blobs). However, only three of the roles (Judge, Attorney, and Prosecutor) can have full access to the connection for sending and receiving all media and data types, which they may use to exchange trial-related documents. In turn, a user playing the role of Defendant may only send and receive audio and video. Finally, the second connection aims at supporting a private audio/video conversation between the attorney and her defendant.

4.3. RBCML runtime model

Once a model is instantiated, a runtime model represents the runtime entities that constitute the resulting communication session. The metamodel used to define runtime models is a direct extension of RBCML's metamodel, as shown in Figure 3. The changes are meant to represent the actual users taking part in a session (and fulfilling the roles specified in the respective RBCML model), along with the actual devices that they are currently using to connect. The remaining elements of the RBCML metamodel stay the same. Note that, although a role may be fulfilled by more than one user in the same session (according to the minimum and maximum cardinalities specified in `RoleConnection`), a given

```
"session":{
  "roles":["role":{"idrole":"1","name":"Judge"},
           "role":{"idrole":"2","name":"Defendant"},
           "role":{"idrole":"3","name":"Attorney"},
           "role":{"idrole":"4","name":"Prosecutor"}, {...}],
  "connections":[
    "connection":{"idconnection":"1","name":"Court Trial",
       "form":{"typedata":["anyType", "stringType"]},
       "media":{"Audio":"true","Video":"true"},
       "attachedRoles":[
           "attachedRole":{"idrole":{"1","3","4"},
              "capabilities":{
                 "send":["Audio","Video","Blob","String"],
                 "receive":["Audio","Video","Blob","String"]}},
           "attachedRole":{"idrole":"2",
              "capabilities":{ "send":["Audio","Video"], "receive":["Audio","Video"]}}
           "attachedRole":{...}, ...]},
    "connection":{"idconnection":"2","name":"Private Defense Conference",
       "media":{"Audio":"true","Video":"true"},
       "attachedRoles":[
          "attachedRole":{"idrole":"2",
             "capabilities":{ "send":["Audio","Video"], "receive":["Audio","Video"]}},
          "attachedRole":{"idrole":"3",
             "capabilities":{"send":["Audio","Video"], "receive":["Audio","Video"]}}]}]
}
```

Figure 2. Sample communication model in J-RBCML.

user can only play a single role in a session. Nevertheless, by fulfilling a role, a user can participate in more than one connection in the session. Furthermore, at a given moment, a user may participate in a session by using only a single device. However, the user can switch devices without disrupting her participation in the session, thus providing essential support for ubiquitous computing.

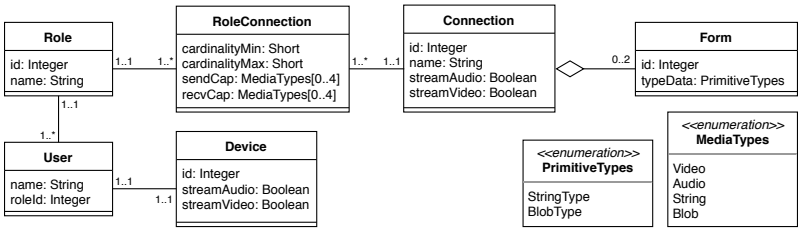

Figure 3. Metamodel for runtime models.

The runtime model serves the purpose of enabling inspection of the current configuration of a communication session, e.g., to obtain the users currently connected and their respective roles and constraints. It can be exposed either in a programmatic way or in the form of a dashboard. However, as future work, we plan to use the runtime model as the basis for a causally connected representation of communication sessions, such that the properties of a session can be changed at runtime by manipulating its model. This can be realized using the mechanisms described by [Bennaceur et al. 2014].

5. Implementation

RBCML is meant to be implemented by translating the elements of communication models into calls to the API of an underlying communication platform. In this section, we describe a prototype in which such translation is carried out using model-to-code transformation, before actual model execution. RBCML can, in principle, be realized on top of any communication platform. In the prototype, however, we chose to implement it on top of WebRTC [Sredojev et al. 2015], which provides a comprehensive API for setting up and controlling direct connections between Web-based applications for the transfer of live streams of audio, video and data. While the programming model of WebRTC does

not directly support the high-level abstractions of RBCML (notably session, role and media constraints), as described below those abstractions can effectively be built on top of the simpler building blocks that it provides. Moreover, the fact that WebRTC is widely deployed (most mainstream Web browsers implement it) makes it an attractive choice to build the type of user-centric communication capability that RBCML provides.

In the prototype, model-to-code transformation is performed on-the-fly, typically as communication models are submitted for execution. The result is the generation of application code that corresponds to the communication structures specified in the model. Figure 4 presents an overview of the code generation process.

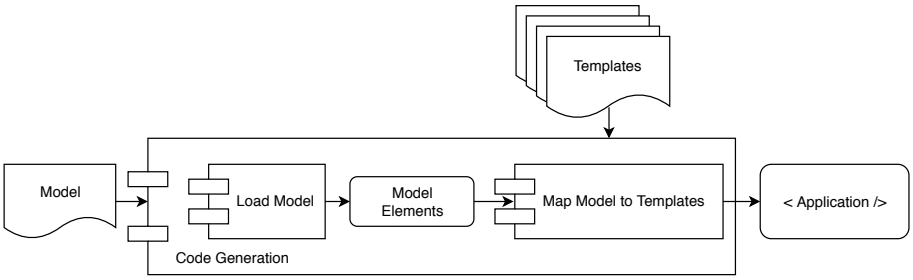

Figure 4. Model-to-code transformation.

The user provides, as input, a suitable model for the communication session. This model may be created first-hand or obtained from a repository of reusable models. The model is loaded and parsed to isolate its elements (e.g., connections, user roles, device capabilities, media streams, and file transfers). The identified model elements are then mapped into their corresponding code templates. The templates, in turn, are completed with information from the communication model (by providing values to parameters such as role IDs and cardinalities) and concatenated according to the structure defined in the model. Note that this process encompasses both the structure of the communication session (the connections and their endpoints) and the layout (i.e., the user interface) of the Web application. The latter is generated using the AngularJS framework, parameterising a generic HTML layout with specific information from the model. As future work, we intend to separate these two aspects of code generation, enabling different customised layouts for the same application.

The generated JavaScript application is made available on a Web-based repository, with a well-known URL. Users who intend to join a communication session must first point their browsers to that URL, loading a list of available communication sessions. By clicking on the desired session's link, the corresponding generated application is loaded on the user's browser. The user then provides his/her ID and requests to join the session. The user's ID is matched with the roles described in the communication model of the session to determine which session connections must be established for that user.

At this point, WebRTC signalling takes place between each peer intending to join a session and a signalling server. Signalling involves a series of message exchanges using the Session Description Protocol (SDP) so that each peer joining a connection receives the relevant connection parameters. Note that, according to the WebRTC protocol, the signalling process must be initiated by the peers themselves. Therefore, a connection (and indeed the RBCML session that contains it) is created as the first peer (which in

principle may be any of the involved users, henceforth called *Peer 1*) decides to start communication. *Peer 1* sends a message to the signaling server, which in turn creates a connection (a *room* in WebRTC jargon) and places *Peer 1* in it. The signalling server then sends back a message to *Peer 1* informing that the room (i.e., the connection) has been created and confirming that *Peer 1* is the initiator. This process is repeated for each connection in which *Peer 1* participates according to the model. Further peers joining the session send their connection information to the signalling server and, according to the connections specified in the model for their respective roles, receive the corresponding connection information from all the other peers already in those connections.

Each peer in a connection must capture its local streams as usual in WebRTC. However, at runtime, just before actual communication starts, any constraints expressed in the model related to sending and receiving media must be applied to the configuration of the connections. In this way, it is possible to disable the sending and/or receiving of audio/video by some endpoints that play a given role. As WebRTC does not support such a feature, it is achieved by directly editing, during signalling, the SDP messages that describe the streams in each connection. The signalling server delegates this task to a component called ApplyConstraints, which edits the media description fields on the SDP message received from a peer before forwarding the message to the other peers in the connection. Figure 5 illustrates this process for a connection with two peers.

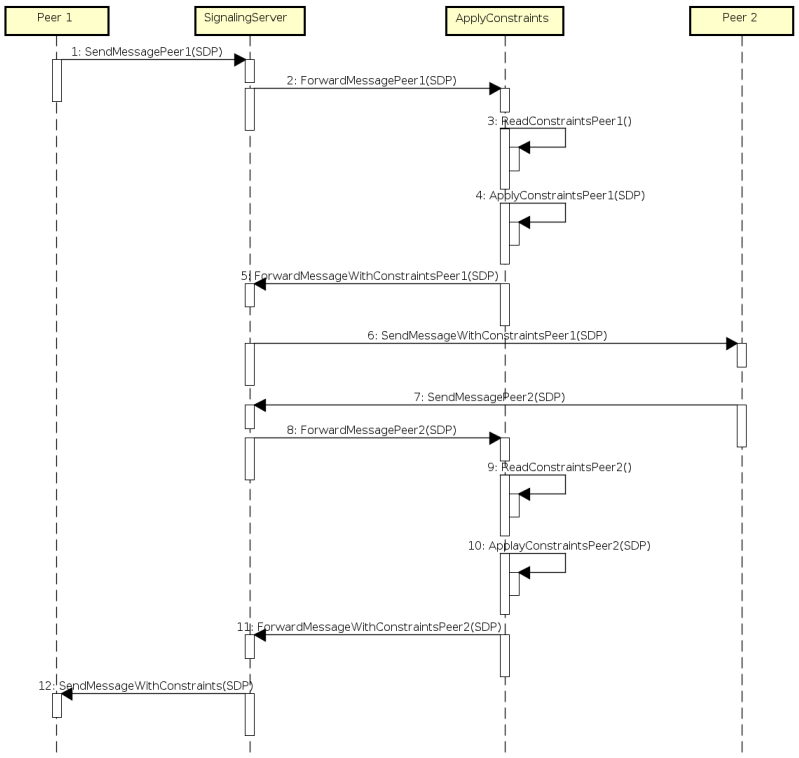

Figure 5. Application of media constraints.

6. Evaluation

Evaluation of RBCML and its implementation was performed both from a usability perspective, through experiments with real users, and from a performance perspective, by

assessing the impact of model processing on application performance. In this section, we describe these two experiments and their results.

6.1. Usability evaluation of RBCML

We assessed the usability of the modelling language using a small controlled experiment involving eight users. The users were volunteers from the undergraduate and Masters programs on Computer Science and Information Systems at UFG. The experiment was divided into two parts. First, the students were asked to implement a simplified communication session using JavaScript and HTML5. Next, they repeated the exercise using RBCML (more specifically, its JSON-based syntax, J-RBCML).

To mitigate any bias related to the fact that the same group of students took part in both cohorts of the experiment, we carried out a survey and extensive training on both technologies before actually starting the assignment. First, the students answered a questionnaire to assess their background. The results showed some heterogeneity in the group, also evidencing that most of the students had no substantial knowledge of JavaScript and HTML5. Thus, we conducted training sessions on JavaScript and HTML5, followed by training on J-RBCML. Likewise, the students also received training on the concepts involved in the design of the application. In this way, we ensured that they were familiarised with both the technologies and the application design from the start so that carrying out the assignment using one technology first would not significantly further their familiarity in a way that would artificially favour the evaluation of the second technology.

The application developed in the experiment had the simple requirement of establishing a two-party communication session, in which the two connected users should have the ability to send and receive audio and video streams. The assignment was conducted in groups of two students using the *pair programming* methodology [Saltz and Shamshurin 2017]. In each group, while one student was the *driver* (programmer), the other was the *navigator* (looking for errors and planning the overall structure of the code). They alternated between these two roles during the assignment. The use of this methodology helped further reduce the effect of the heterogeneity of the groups.

We measured the time taken by each group to develop the two versions of the application. As shown in Table 1, significant productivity gains (60% in the worst case) were achieved with RBCML, when compared with programming in JavaScript/HTML5.

	Group 1	Group 2	Group 3	Group 4
RBCML	27	18	19	23
JavaScript/HTML	68	72	63	–

Table 1. Development time comparison for a simple two-party session (in min)

Finally, after completing the assignment, a second questionnaire was applied, individually to each participant, to evaluate the difficulties and issues they experienced while developing both versions of the application. Out of the four groups, three were able to successfully finish the assignment both in JavaScript and in RBCML, while one group was able to complete only the RBCML part. From the answers to the questionnaire, it became evident that RBCML, with its model-based approach, has a much higher level of objectiveness and appropriateness to the domain than conventional programming

in JavaScript. The students made several remarks about the intuitiveness and abstraction power of RBCML, as it hides most of the complexities of application development while still producing equivalent results. The main difficulty reported by the students with the modelling approach was related to their lack of previous knowledge on JSON and the J-RBCML syntax. Notice, however, that J-RBCML is meant mainly for internal model representation. We, therefore, expect to eliminate such difficulties with a complete implementation of the toolchain, notably including support for the graphical syntax for RBCML, in the form of an integrated modelling environment.

Note we should take these results with caution, considering them as initial indicators of the promising qualities of the proposed approach. We acknowledge the threats to the validity of the experiment, notably the size of the sample (only 4 groups of 2 students) and the fact that the same groups worked on the assignments using both technologies and in the same order. In the future, we aim to repeat the experiment using a larger sample, with different groups working with each technology. Moreover, to be more realistic, this should be carried out using the graphical syntax of RBCML.

6.2. Performance of model processing

Table 2 shows the results of a performance evaluation carried out to assess the impact of RBCML model processing over the time required to establish communication sessions. We measured session establishment time both *with* and *without* the use of RBCML. For the former, we used three slight variations (based on the number of media constraints that need to be applied to the connections) of the two-party session model described in Section 6.1. In contrast, for the latter, we used (as baseline) a functionally equivalent application directly developed in JavaScript. The different steps of session establishment (columns 2–8) are shown separately to characterize the model processing overhead more precisely. The experiment was carried out using three 2.4GHz Intel Core i3 machines with 4 cores, 8GB RAM, running Linux Mint 18.2. Two machines were used to host the browsers (Chrome release 71), which in turn hosted the session peers, while the third machine was used to host the signalling and model processing server. A Python script was used to emulate the users at each host, and the times presented in each row represent the average of 500 executions of the experiment. For each execution, the server was started from scratch to force the execution of all steps (including code generation – although this is a worst-case scenario, as generated code can be reused from previous deployments of the same model). Please note that network times were not measured since RBCML does not require additional messages (compared to standard WebRTC-based applications) nor any extra overheads on the standard WebRTC messages.

As the table shows, the overhead of RBCML model processing corresponds to model loading (col. 2), code generation (col. 3), and media constraint application (cols. 7 and 8). The first two are roughly constant for a given model size (8 elements in this experiment), i.e., it does not vary with the number of constraints. Constraint application time rises with the number of constraints, noting that this overhead exists even if there are no constraints to apply (since the model needs to be inspected for the existence of constraints in any case). On the other hand, application loading has an insignificant overhead, and the creation of peer connections do not suffer from any overhead at all, as can be seen from the comparison with the application built without using models (last row). The total overhead for a two-party session model, as can be seen in the last column, is

	Load Model	Generate Code	Load App	Peer Conn 1	Peer Conn 2	Apply Constraints Peer 1	Apply Constraints Peer 2	Total
Without media constraints	0.234	0.397	0.375	0.100	0.100	0.240	0.240	1.686
With 1 media constraint on Peer 1	0.235	0.398	0.374	0.100	0.099	0.310	0.240	1.756
With 2 media constraints on Peer 1	0.237	0.398	0.374	0.100	0.100	0.336	0.240	1.785
Baseline (app developed in JavaScript	–	–	0.353	0.099	0.100	–	–	0.552

Table 2. Breakdown of session establishment time (in ms).

1.2ms. This overhead grows with the number of peers, n, with a factor $n(n-1)$, which represents the number of peer connections to be created, and thus the number of times the *apply constraints* step needs to be executed. Nevertheless, as communication applications are typically bound by user interaction, and as the number of peers tends to be reasonably small, overheads in the order of a few milliseconds are not perceptible. Furthermore, note that the performance during session operation is not affected by the use of models, since, once established, a session's connections are just standard WebRTC connections.

7. Concluding remarks

In this paper, we presented RBCML, a communication modelling language based on user roles, which enables the creation of communication session models consisting of multiple connections with different media features. Such communication models can be reused in different scenarios, with a varying number of users, according to the cardinality of their corresponding roles. We described a prototype built on top of the well-established WebRTC platform. We demonstrated, through an experiment with real users, that the use of RBCML dramatically reduces the effort required to create such applications, as compared to WebRTC applications natively developed in JavaScript. We also carried out an experiment to show that the performance of applications developed using RBCML is not significantly lower than similar applications developed without models. As future work, we plan to develop the entire RBCML toolchain, including graphical modelling, which shall further improve usability. We also aim to validate the use of RBCML in ubiquitous computing scenarios, in which users can seamlessly change devices without disrupting their participation in ongoing sessions. Finally, we need to investigate the security issues involved, notably to ensure that only authorised users participate in communication sessions (and taking up the correct roles). It is also essential to ensure the enforcement of constraints on session connections, as malicious users may tamper with SDP messages to overcome the stream constraints expressed in the model. Running SDP over IPSec or TLS would be a starting point to investigate these issues.

Acknowledgements

The authors would like to thank FAPEG, Brazil, and CNRS, France, for partly funding the work presented in this paper.

References

Bennaceur, A., France, R., Tamburrelli, G., Vogel, T., Mosterman, P. J., Cazzola, W., Costa, F. M., Pierantonio, A., Tichy, M., Akşit, M., Emmanuelson, P., Gang, H., Georgantas, N., and Redlich, D. (2014). Mechanisms for leveraging models at runtime in self-adaptive software. In Bencomo, N., France, R., Cheng, B. H. C., and Aßmann, U., editors, *Models@run.time: Foundations, Applications, and Roadmaps*, pages 19–46. Springer International Publishing, Cham.

Bézivin, J. (2005). On the unification power of models. *Software & Systems Modeling*, 4(2):171–188.

Cho, K., Fukuda, K., Esaki, H., and Kato, A. (2006). The impact and implications of the growth in residential user-to-user traffic. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 207–218. ACM.

Clarke, P. J., Hristidis, V., Wang, Y., Prabakar, N., and Deng, Y. (2006). A declarative approach for specifying user-centric communication. In *International Symposium on Collaborative Technologies and Systems (CTS'06)*, pages 89–98.

Contreras, F. R., Alvarez, B., Sanchez, P., and Pastor, J. A. (2016). U-DSL: A domain specific language for ubiquitous systems. *IEEE Latin America Transactions*, 14(10):4416–4420.

de Azevedo Vieira, M. B. (2018). Uma abordagem dirigida por modelos para comunicação em tempo real. Master's thesis, Instituto de Informática, Universidade Federal de Goiás, Goiânia-GO, Brazil. (in Portuguese).

de Farias, C. R., Leite, M. M., Calvi, C. Z., Pessoa, R. M., et al. (2007). A MOF metamodel for the development of context-aware mobile applications. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 947–952. ACM.

ITU-T (2000). ITU-T Recommendation F.703 - Multimedia Conversational Services. Standard ITU-T F.703, International Telecommunications Union - Telecommunication Standardisation Sector, Geneva, CH.

Jia, Q., Xie, R., Huang, T., Liu, J., and Liu, Y. (2017). The collaboration for content delivery and network infrastructures: A survey. *IEEE Access*, 5:18088–18106.

Medvidovic, N. and Taylor, R. N. (1997). A framework for classifying and comparing architecture description languages. *ACM SIGSOFT Software Engineering Notes*, 22(6):60–76.

Saltz, J. S. and Shamshurin, I. (2017). Does pair programming work in a data science context? an initial case study. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2348–2354.

Sredojev, B., Samardzija, D., and Posarac, D. (2015). WebRTC technology overview and signaling solution design and implementation. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1006–1009.

Tindall, N. and Harwood, A. (2015). Peer-to-peer between browsers: Cyclon protocol over WebRTC. In *2015 IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 1–5.