PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 12/13

FSMDA Instantiation to Support NCL 3.1 Applications in Ginga Reference Implementation

Luiz Fernando Gomes Soares
Carlos Eduardo Coelho Freire Batista

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 12/13 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena December/2013

FSMDA Instantiation to Support NCL 3.1 Applications in Ginga Reference Implementation

Luiz Fernando Gomes Soares
Carlos Eduardo Coelho Freire Batista

Laboratório TeleMídia DI – PUC-Rio
Rua Marquês de São Vicente, 225, Rio de Janeiro, RJ - 22451-900.

lfgs@inf.puc-rio.br
bidu@telemidia.puc-rio.br

Abstract. *Applications that are somehow related to the content presented on the main screen have been introduced to collective spaces, such as rooms with Digital TV, theaters with Digital Cinema, conferences with interactive presentations, etc. In this technical report we introduce a framework to support the execution of distributed multi-device applications aiming at these collective spaces. The main contribution is an interoperable software platform that offers high-level abstractions that hide or minimize the complexity of dealing with this distributed and heterogeneous execution environment, unlike existing solutions that have limited functionality or are constrained to device brands. The proposed architecture is presented by means of its instantiation, part of middleware Ginga, and the support offered by the NCL language. However, the framework is not constrained to NCL and Ginga. The instantiated platform has been tested against different scenarios, which are also presented.*

Keywords: *multi-device applications, companion screen, digital TV; public display, digital cinema, middleware; declarative environment; NCL.*

Resumo. *Aplicações que estão de alguma forma relacionadas com o conteúdo apresentado na tela principal têm despontado em espaços de exibição coletivos, tais como salas com TV digital, cinema digital, conferências com apresentações interativas etc. Neste relatório técnico, nós apresentados um "framework" para suporte à execução de aplicações distribuídas em múltiplos dispositivos, tendo como alvo esses espações coletivos. A contribuição principal é uma plataforma de software interoperável que oferece abstrações de alto nível que escondem ou minimiza a complexidade do tratamento deste ambiente de execução distribuído e heterogêneo, ao contrário das soluções existentes que possuem uma funcionalidade limitada ou estão restritas a produtos específicos. A arquitetura proposta é descrita por meio de sua instanciação, parte do middleware Ginga e do suporte oferecido pela linguagem NCL. No entanto, o "framework" não está restrito ao Ginga e à NCL. A plataforma instanciada foi testada frente a diferentes cenários, que também são apresentados.*

Palavras chave. *aplicações multi-dispositivos, tela companheira, TV digital, cinema digital, middleware, ambiente declarativo NCL.*

FSMDA Instantiation to Support NCL 3.1 Applications in Ginga Reference Implementation

Laboratório TeleMídia

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rua Marquês de São Vicente, 225, Prédio ITS - Gávea

22451-900 – Rio de Janeiro – RJ – Brasil

http://www.telemidia.puc-rio.br/

Table of Contents

FSMDA Instantiation to Support NCL 3.1 Applications in Ginga Reference Implementation

Luiz Fernando Gomes Soares
Carlos Eduardo Freire Batista

Laboratório TeleMídia DI – PUC-Rio
Rua Marquês de São Vicente, 225, Rio de Janeiro, RJ - 22451-900.

lfgs@inf.puc-rio.br
bidu@telemidia.puc-rio.br

Abstract. *Applications that are somehow related to the content presented on the main screen have been introduced to collective spaces, such as rooms with Digital TV, theaters with Digital Cinema, conferences with interactive presentations, etc. In this technical report we introduce a framework to support the execution of distributed multi-device applications aiming at these collective spaces. The main contribution is an interoperable software platform that offers high-level abstractions that hide or minimize the complexity of dealing with this distributed and heterogeneous execution environment, unlike existing solutions that have limited functionality or are constrained to device brands. The proposed architecture is presented by means of its instantiation, part of middleware Ginga, and the support offered by the NCL language. However, the framework is not constrained to NCL and Ginga. The instantiated platform has been tested against different scenarios, which are also presented.*

1. INTRODUCTION

More and more consumers are using another device connected to the internet to check emails, texts, browse on the web, shop, relate on a social network, etc., while watching television, slideshow presentations, etc. Digital multitasking while watching some video content has clearly become common usage.

In particular, a fundamental transformation is taking place in the "lean back" home video entertainment environment. The passive TV viewing experience is giving way to a far more interactive one. This transformation has begun in the beginning of the last decade with the deployment of interactive applications in terrestrial digital TV systems [1, 2, 12] and has gained impulse in the last years both with the launching of television sets with integrated Internet capabilities and the introduction of companion screen applications. Today, rather than solely broadcast-based, the new TV model is increasingly a broadcast/Internet based hybrid [13, 20, 21, 27], and, rather than passive, it is increasingly interactive.

Companion screen applications (also named second screen applications) refer to TV-related applications specifically designed to be used while watching TV on the main screen. Applications that are somehow related to the content presented on the main screen have also been introduced to other collective spaces, such as theaters with Digital Cinema, conferences with interactive presentations, rooms with public display, etc. In these applications, secondary devices – such as smart phones, tablets, and laptops – are used to search, interact and engage with a rich variety of applications, websites and communities while watching the main screen. Even though some of these applications can be used on their own, they are usually aware of the content displayed on the main screen (TV most of the time).

Although the great majority of TV-related applications are screen based, in some of them secondary devices are not limited to ones that allow for second visible apparatus. As an example, the next section presents an interactive movie narrative whose content flow is controlled by viewers, randomly chosen in the movie theater, by using a phone call, that is, using an audio device. Moreover, applications can run with independent control on each secondary device, but can also run under a single control logic distributed over secondary devices and the device that provides the main screen. So, to be more general, we will call these applications "multi-device applications".

In this technical report we are not interested in multitasking in its broad sense, but only in applications that are somehow related to the content presented on the main screen: a TV set, a movie screen, etc. More precisely, we are interested in applications targeting multiple devices, which are specifically designed to be used while watching on the main screen. We are interested in offering a framework to support this kind of applications and its use in Ginga reference implementation to support multi-device NCL applications.

Several proprietary solutions have been deployed to support multi-device applications, but the real problem comes when these applications run on several heterogeneous devices that must communicate with each other (including the device supporting the main screen) to accomplish a distributed synchronous presentation, mainly under a single logic control, but with commands coming from multiple viewers. In this case, interoperability is a very important issue; reuse of software components among applications is crucial; fault-tolerant procedures are required, and support to current and upcoming applications is very desirable in this application field that has just taken the first steps.

This technical report presents a framework (FSMDA – Framework to Support Multi-Device Applications) that can be customized to any middleware standard [1, 2, 12, 13, 21], including a multi-standard customization, and presentation description language (NCL-Lua; HTML-JavaScript, SVG, SMIL, etc.). FSMDA runs distributed hypermedia applications integrating resources from heterogeneous interconnected devices, considering interoperability and fault-tolerant requirements. Depending on the FSMDA instantiation, it can support each individual set of multi-device applications already reported and summarized in Section 2, or all of them, including those to each there is not yet a recognized conceptual model behind. FSMDA is presented based on its instantiation targeting support to NCL (Nested Context Language) applications, extending its middleware called Ginga, ITU-T Recommendation for IPTV services [21] and terrestrial digital TV [1]. The new version of NCL and Ginga have been recently submitted to ITU-T to be incorporated in the next version of the H.761 Recommendation. In this technical report we also discuss some system test case developed for three scenarios targeting the three collective spaces aforementioned.

The technical report is organized as follow. Section 2 analyzes several multi-device applications highlighting its characteristics and differences, trying to establish which scenarios the FSMDA proposal should support. Based on Section 2, Section 3 defines some non-functional requirements for FSMDA. Section 4 discusses current approaches with similar intent, in order to highlight our contribution. Taking NCL as a use case, Section 5 presents the data structures introduced by FSMDA to support multi-device applications. Section 6 details the proposed software architecture introducing its instantiation, adopted by the Ginga middleware reference implementation. Conducted system test cases are presented in Section 7. Finally, Section 8 presents some final considerations and summarizes some future work.

2. BACKGROUND

First, let us define (and redefine) some terms used in our discussion:

- Main device: the device that runs the code (a module of the distributed multi-device application or an independent application) that controls the presentation on the main rendering apparatus (e.g., the main screen);
- Secondary device: any device used to run a code (a module of the distributed multi-device application or an independent application) while the main device is controlling the main presentation;
- Distributed multi-device application: an application with modules that run on the secondary devices and on the main device, in which there are some sort of communication between these modules to perform the distributed presentation;
- Main application: an independent application or a *module of a distributed multi-device application* that controls the presentation on the main device;
- Main audiovisual content: content presented on the main device (usually a stream or a slide show) that multi-device applications have as a target; the main audiovisual content at least starts its presentation on the main device and can be either the single content in presentation or be presented with other content on the main device, under control of the main application;
- Companion application: an independent application or a *module of a distributed multi-device application* that runs in parallel with the presentation on the main device;
- Companion screen application (or second screen application): a particular case of companion application in which its running code render some content in the screen of its secondary device;
- Independent application: the main application or a Companion application that does not communicate with other devices.

We have analyzed more than a hundred commercial multi-device applications, including companion screen applications discussed and presented in [29], [34], and those focusing on social inclusion designed to be used by the Brazilian profile of the ISDB-T digital TV [1]. Most of applications target the digital television domain, but some of them also target Digital Cinema and Slideshow presentations.

There are several taxonomies in the literature grouping multi-device applications with regard to their purposes, to their functionalities. In this technical report we follow a non-functional approach, classifying these applications according to:

- their control logic: distributed with single control logic, or distributed with multiple control logic;
- the scope of the main application: if it controls the display just of the main audiovisual content on the screen, or if it controls additional content presented on the main device;
- their presentation: on single or on multiple devices;
- the semantic relationship between the content presented on the main device with content presented on secondary devices: related or not related to the content meaning presented on the main device;
- the temporal and spatial synchronization relationship between the content presented on the main device with content presented on secondary devices;
- the communication established between the main device and secondary devices, and between secondary devices, in the case of distributed applications.

Independent main applications displayed on a single screen are usual in broadcasted DTV applications. Some of them are semantically related with the main audiovisual content, and some are not, but the majority does not have any temporal synchronization with the main audiovisual content, except that they must start during the presentation of this content. As examples, Figure 1 presents a health broadcasted DTV application with neither semantic nor synchronization relationship with the main video stream; Figure 2 shows an advert application synchronized with the main video: at the exact moment the building with apartments to sell appears in the main video, the picture with the building localization is presented.

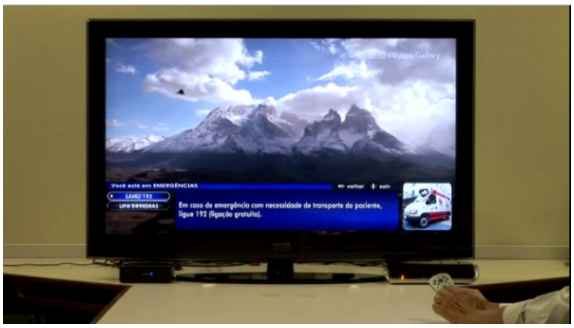


Figure 1 - T-health application about dengue fever

Figure 2 - TV advert application: apartment sale

Of course independent main applications displayed on a single screen are out of this technical report concern. Some simple distributed multi-device applications only offer a kind of a more sophisticated remote control. Companion devices are used only to send commands to the main device, taking profit of its interaction advantages such as touch screens, sensors and high capable graphics rendering. The companion device can be used to launch, pause, play, hide, resize, fast forward or rewind the main audiovisual content. In these distributed multi-device applications there is no communication between secondary devices. The communication is only between the secondary devices and the main device; a unidirectional communication only to send commands.

Figure 3 shows the AT&T remote control. Once the content has been selected and launched on the main screen, the companion screen can still be used to pause, play, fast forward or rewind the content. Another example is Chromecast (Figure 4) which allows for browsing for what to watch, control playback, and adjust volume using a secondary device.

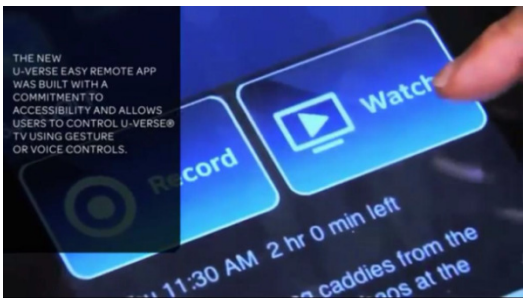


Figure 3 – Using AT&T remote control application
(figure took from [29])

Figure 4 – ChromeCast
(https://www.google.com.br/intl/en/chrome/devices/chromecast/)

TV discovery applications are, in general, examples of distributed multi-device applications with very similar characteristics to the last previous group of applications. TV discovery

applications allow searching, selecting and launching the selected content from the companion screen to the main screen. Figure 5 shows an example of this kind of application.

Figure 5 - SkyPlus iPad application EPG with several categories to sort programs: Entertainment, lifestyle, movies, favorites (figure took from [29])

Distributed multi-device applications without any communication among secondary devices and in which secondary devices communicate with the main device only to send commands are also out of this technical report concern.

Still very closed to the independent main applications, we have the distributed multi-device applications with single control logic to present their content on multiples devices. Secondary devices must register on the parent device running the application logic in order to be able to present the content addressed to them. We say that they run in a *passive class*. Figure 6 shows a tic-tac-toe game as an example. The application runs on the main screen device during a Kids Show. The game is displayed both on the main and companion screens. Although semantically related with the main screen's content, there is no temporal synchronization with this content, except that the application is enable only during the Kids show. A bidirectional communication between the (parent) main device and the secondary devices must be established. Secondary devices communicate with each other via the parent device.

Figure 6 – Tic-tac-toe game running on the primary device (first screen) and displayed both on the first and companion screens.

Social TV applications aim at creating an attractive social TV network, using TV content as a driver to join the network. They are examples of distributed multi-device applications with independent control in which communication between secondary devices is essential. Social TV usually consists of conversations about TV shows, carried out on social networks, to rate, recommend or discuss shows—during or after initial broadcasting. Social TV is without doubt the most popular way to use the companion screen. Most of commercial Social TV applications do not have any synchronization with the main screen content. Communication with the main device is not usual, except to start an application by using some ACR (Authomatic Content Recognition) technology [3, 10]. Figure 7 present an example on commenting TV shows. SideCastr application allows for synchronizing the main audiovisual stream on the main screen with the companion screen for some specific TV shows that are listed in the application. Then the consumer can comment in some pre-defined categories and see other people's comments while the show is on. In the example, SideCastr recognized automatically that the NBA final 2012 was playing. In [29] several other Social TV application examples (ConnectTV, tvtag, IntoNow, WayIn, etc.) can be found.

Figure 7 – SideCastr application: comments can be post in any pre-defined categories (figure took from http://www.sidecastr.com/).

In Social TV applications, secondary devices communicate with each other via a social networking service, usually a Web-based service. The communication between the main device and secondary devices is unidirectional, when available, to allow for the main application sending synchronization commands to companion screen applications. The communication is performed by some broadcast protocol, or via some ACR technology, using the main audio (watermarking and fingerprinting) or visual information placed on the main screen (QR code, for example) for synchronization.

Enrichment applications aim at giving extra information or content about the main audiovisual content, to allow interactivity and enhance customer experience, especially while watching the main screen. The additional content can be available at different moment of the timeline: content available all the time; specific content available only before or after synchronized content; content synchronized with the main screen available only during the show. The type of interaction — synchronized with the content on screen (whether the content is a live broadcast or recorded and watched at a later time), or complementary (independent of the broadcast, which can be used even without the main screen) — will determine, the appropriate automatic content recognition (ACR) system or communication protocol required. As an example of Enrichment applications with independent control, IntoNow companion screen application recognizes automatically which TV show/channel is played on the main screen and displays automatically related information. In this example, ABC World News was live and the screenshot in Figure 8 shows informational content about the American election, allowing access to even further information by tapping on "who will win" or the electoral scoreboard.

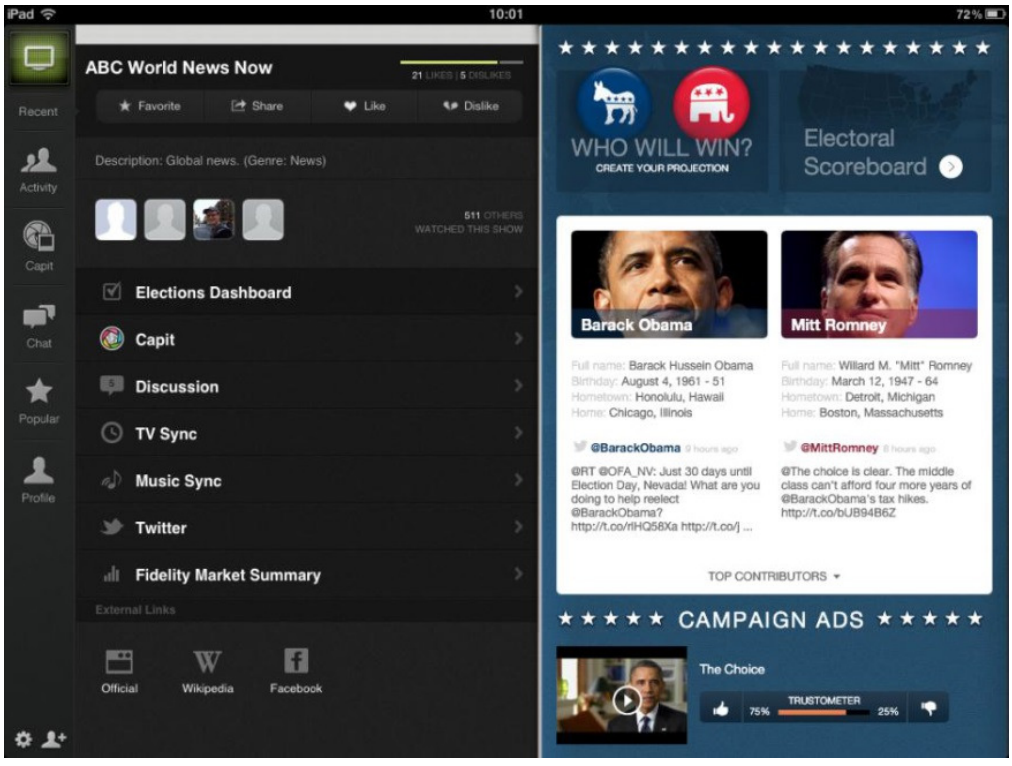


Figure 8 – Providing additional content automatically with IntoNow (figure took from [29])

IntoNow is an example of Enrichment application in which the communication between the main device and secondary devices is unidirectional, to allow for the main application

sending synchronization commands to companion screen applications. The additional information presented on secondary devices is provided by some Web service.

Enrichment applications are well known applications in terrestrial DTV systems. There are a lot of independent main applications designed for these platforms. Figure 2 shows a TV commercial example. A study led by Decipher for Red Bee Media has shown that more people are waiting for companion screen applications for live events than for other content [9]. As another example of Enrichment application with independent control, the companion screen application in Figure 9 allows for buying tickets for the FIFA World Cup 2014 and is synchronized with the main screen soccer game using QR Code.

Figure 9 – QR code launching a companion screen application

In terrestrial DTV it is usual having the additional companion screen content received from the main application. The main application is in charge of receiving both the content to be presented on the main device as well content to be presented on secondary devices. ACR technology is not used in this case. Instead a bidirectional communication channel is established between the main device and the secondary devices, using some pre-defined protocol. Secondary devices must pair to the main device to receive the content to be presented and synchronization commands. Sometimes, even the companion application logic control code is received from the main device. The communication channel is bidirectional because it is usual having the companion application also sending synchronization command to the main application. As an example, Figure 10 shows a snapshot of a companion screen Enrichment application controlled by the Ginga middleware [39], which is responsible for synchronizing the additional interactive information with the content displayed on the main screen. The application is related to the Band News live broadcast. The content shown on the companion screen was broadcasted, received by the main device (a TV set with Ginga middleware) and sent to the companion screen device via Wi-Fi or Bluetooth network.

Figure 10 – Companion screen of the Band News enrichment application

The Gamification concept has been adopted by some companion screen applications. It consists in rewarding users when they perform some specific actions. Figure 11 shows a distributed multi-device application under the control of the Ginga middleware [39], which is responsible for synchronizing the puzzle game information with the monument displayed on the main screen. For each monument there is a puzzle that gives points to viewers that manage to complete it. The points can be then used to earn discounts in an online bookstore. This application has exactly the same non-functional characteristics presented for the application shown in Figure 10.

Figure 11 – Increasing engagement with gamification.

As a last example of Enrichment applications, Figure 12 shows a soccer match in which, synchronized with moment that a penalty is signalized, a game is sent to the two first devices paired to the main device (the “penalty shooter” is sent to the first device and the “goalkeeper” to the second one). The received game prompts their viewers for choosing a direction. Afterwards, an animation exhibits the result of the penalty shootout on the main screen.

Figure 12 – Brazilian movie invitation to the 2014 FIFA World Cup

The non-functional requirements of this application are particularly interesting. First, we must establish a bidirectional communication channel between the main device and the secondary devices. Secondary devices must pair to the main device to establish the connection. Only the first and second paired device will receive the game from the main application. The game is adapted to the viewer role: shooter or goal keeper. The choices made by the first two paired devices are sent back to the main application that will show the game result on the main screen. We can see here the logic of the main application inducing the logic of the companion screen application and vice versa, by exchanging command and media content.

For over a decade, producers have been looking for the most effective ways to marry TV to the Web. Some criticism argues that when bringing the Web's content model to the TV we are limiting the possibilities. Distributed multi-device applications, with a rich synchronize communication between the multiple devices bring new content models, including interactive narratives. Two examples are shown n in Figures 13 and 14. In the first, an interactive zombie movie adventure makes the merchandize of Hell Pizza, a New Zealand-based pizza chain. The movie originally produced for the main screen[1] was modified to run on companion screens. The interactive narrative computes the total of viewers' choice and change the narrative flow according to the majority of votes. Figure 13 shows a decision point in which viewers must decide if the pizza delivery guy lets the man get in the car.

[1] https://www.youtube.com/watch?v=9p1yBlV7Ges

a) Main screen

b) Companion screen

Figure 13 – Hell Pizza interactive narrative[2]

In Figure 14 the secondary devices are mobile phones. Last Call[3] is the first interactive horror movie in the world where the audience is able to communicate with the protagonist. A film is controlled by a member of the audience, thus blurring the boundaries between game and film. Language recognition software transforms the participant's answers via mobile phone into specific instructions that launches an appropriate follow-up scene. To participate in the adventure, audience members submit their mobile phone numbers to a speed dial code when they buy their ticket. The moment the female protagonist takes out her phone to call someone who might be able to help her, the film's controlling software contacts one of the submitted mobile phone numbers. Every single choice shapes her fate: it's a matter of life and death.

a) Main screen (What should I do?)

b) Companion device: mobile phone (Keep going!)

Figure 14 – Last Call interactive narrative

Table 1 summarizes the non-functional analyses of the sixteen applications analyzed in this Section.

[2] A a LITTLESISTERFILMS production. Starring Dj Iwikau, Ben Edwards & Emily Trenberth Producer Katie O'Brien. Writer, Director & Editor Logan McMillan.

[3] 13th Street Last Call is made by Jung von Matt/Spree (Idea), Film Deluxe (Film), nhb (Sound), Telenet (IVR), AixVox and Powerflasher (Software).

Table 1 – Non-functional aspects of the analyzed applications.

Name	Type	Control logic	Communication between secondary devices	Communication between the main device and secondary devices	Data sent from the main app. to companion apps..	Data sent from companion apps. to the main app.	Pairing	Device Management	Fault handling mechanisms	Middleware Support
Kids Show	Game	Single	Via main app.	Proprietary protocol via Wi-Fi or Bluetooth network	All companion device's media content	Interaction notification	Yes Without any verification	As a group	No	Standard DTV middleware architecture with proprietary solutions
SideCastr	Social TV	Distributed	Via social networking service	Proprietary audio fingerprinting	Synchronization information	—	No	As a group	—	Only for ACR
IntoNow	Enrichment	Distributed	Via social networking service	Proprietary audio fingerprinting	Synchronization information	—	No	As a group	—	Only for ACR
FiFA 2014	T-Commerce	Distributed	—	QR code	Synchronization information	—	No	As a group	—	Only for ACR
Band News	Enrichment	Distributed	—	Proprietary protocol via Wi-Fi or Bluetooth network	All companion device's media content, application code and synchronization information	—	Yes Without any verification	As a group	—	Standard DTV middleware architecture with proprietary solutions
Rio Monuments	Gamification	Distributed	—	Proprietary protocol via Wi-Fi or Bluetooth network	All companion device's media content, application code and synchronization information	—	Yes Without any verification	As a group	—	Standard DTV middleware architecture with proprietary solutions
Soccer Match	Enrichment	Distributed	—	Proprietary protocol via Wi-Fi or Bluetooth network	All companion device's media content, application code and synchronization information	Interaction notification	Yes Without any verification	As a group and individually	No	Standard DTV middleware architecture with proprietary solutions
Hell Pizza	Interactive Narrative	Distributed	Via main app.	Proprietary protocol via Wi-Fi or Bluetooth network	All companion device's media content, application code and synchronization information	Interaction notification	Yes Without any verification	As a group	—	—
Last Call	Interactive Narrative	Distributed	Via main app.	Phone call	Audio content	Audio content	Yes Without any verification	As a group and individually	No	—

In Ginga reference implementation we are interested to give support to all kind of distributed multi-device applications in which the main audiovisual content are at least semantically related to information presented on secondary devices. Secondary devices must be aware of the content that is displayed on the main device. Single-device applications are out of discussion in this technical report, since they are already supported by Ginga since its earliest version.

Although there are a number of applications trying to "communicate" with the main application, most of them lack this functionality. The connected ones either have limited functionality or support limited main device brands. Usually applications are non-generic in each specific concept. In other words, used protocols (in terms of communicating with the main device) are brand specific or used by limited solutions. The current discrepancies between platforms and technologies used to produce, activate and optimize multi device initiatives have prompted broadcasters such as the BBC to make an attempt at standardizing companion screen solutions[4]. Interoperability is an important requirement in this technical report proposal. FSMDA architecture tries to be generic enough to be adopted by different standards, or even as a multi-standard solution.

3. REQUIREMENTS

FSMDA is particularly interested in spaces in which one device (the main device) is shared by all users (e.g., a Digital TV receiver, a Cinema projector, a Slideshow projector, a Public Display, etc.), and to which other (possibly heterogeneous) secondary devices can join to support distributed presentations (consumed by a group of people). More precisely, the Ginga instantiation is interested in collective spaces in which a hierarchical resource-orchestration is present, always having the main device as the root of the hierarchical tree, since the hierarchical model is common in several collective spaces.

In all examples discussed in Section 2 we can see only two hierarchical levels: the main application level and the secondary device application level. However there are some cases in which it is important to give support to a depth hierarchical tree. Take as an example slideshow presentation, in which the main screen, connected to a computer (main device), displays the multimedia main content. The main application can delegate control of the presentation to a companion application (second hierarchical level) running on the speaker's secondary device (a tablet, for instance). The speaker may ask questions or send additional content to each person whose mobile device (i.e., a third hierarchical level) is paired with the speaker's secondary device. Results from the interaction between the speaker and the audience may be displayed on the main screen.

We have identified in most of distributed multi-device applications that secondary devices usually act as a group, which we call "class". Earlier [39] we have identified two types of device classes whose functionalities are extended in this technical report. Moreover, two more class types are added in the current proposal. A secondary device can be paired to more than one class offered by a parent device. The four class types can be summarized as follows:

- Passive – devices paired to a passive class must be able to present pre-rendered media streams received from a parent device to which they are paired in the hierarchical

[4] http://tbivision.com/news/2013/03/bbc-calls-for-second-screen-standards/53282/

orchestration tree. They navigate in such content as a group; that is, any navigational action reflects in all paired devices (individual navigation is not allowed).

- Active – devices paired to an active class must be able to decode, render/run and control media content (part of the distributed multi-device application) by themselves, control that has been delegated to them by the parent device to which they are paired. Applications, and their content, delegated to devices in an active class can come from the parent device or from any external mean. In active classes, individual (personalized) navigation on media content is possible. The main device runs this class profile by default and induces the hierarchical orchestration. Each device paired to an active class can also be a parent device and define new classes, hierarchically.
- Media Capture – devices paired to this class can capture media content and send them to the device it is paired to, according to the logic of the multi-device application.
- On Demand Media – devices paired to this class must support the UPnP AV MediaServer ControlPoint service [22], which enables them to browse a video list and request them on demand. The shared video list is controlled according to the logic of the multi-device application.

It should be stressed here that pairing in the scope of class definition can be explicit, by means of some protocol usage, or defined by default. For example, in application of Figure 8, all secondary devices can be considered to be paired by default; the companion screen application was received by an external mean.

Besides the analyzed applications (some of them discussed in Section 2), we have exercised different usage scenarios for four collective spaces (Digital TV, Public Display, Digital Cinema and Interactive Slideshow), using different combinations of device classes. Such scenarios were used to define system test cases presented in Section 7. From our analysis and these scenarios, besides the support to the aforementioned orchestration model, the following high-level requirements were established for FSMDA:

- Coarse (soft) temporal synchronization must be assured among media content being rendered/executed on different devices (fine/hard temporal synchronization should be pursued, but it is not addressed in this technical report);
- Paired devices in a same class may be treated (referred) as a group, or individually;
- Devices must be able to join and leave the application presentation dynamically;
- Communication among devices may use individual or group communication protocols, allowing group and individual media content rendering;
- Heterogeneous information may be collected and sent from parent or child devices;
- Content associated to different device classes may be adapted according to the characteristics of the devices, users, and user locations;
- Interoperability mechanisms must allow for the integration of devices based on different hardware and software, and must support different network services and protocols;
- Automatic and semi-automatic fault handling procedures must be conducted on situations in which components fail or act arbitrarily against the distributed application logic.

4. RELATED WORK

Several empirical studies show a clear tendency of the user to use a second device while watching television [16, 33]. Some other studies [4, 30] show some apparent new behaviors on the part of users when consuming content via multiple devices.

What do people want while they watch TV? According to Miso [30], the more we ask people about what they want, the more we realize that the answer is fundamentally different. No one really has the answer yet. Maybe because not all possibilities have been exposed to and experienced by users who remain trapped to web-based models. Many questions regarding valuable and useful concepts in an interconnected multi-screen environment stay unanswered. That is why it is so important to have a framework that can be able to support upcoming applications and not only the current ones. Trying to understand the current practice of multi-device usage within domestic contexts Huang et al. [19] empirically explored the activity of chatting while watching TV; Geerts et al. [15] report insights on how the program genre of a TV show has influence on the communication behavior of the users; Hess et al. [18] look for user requirements and preferences for an integrated social media system to be applied in future designs and developments. On the other hand, MediaTVcom [29] analyses the segmentation of the companion screen application market and describes the challenges to launch such applications.

The great majority of companion screen applications [29, 34] are distributed client-server applications with independent control running on each secondary device. Some of them, like Discovery applications [29] do not have communication among client secondary devices running the application. In some other, like Social TV applications [29], the client side applications may communicate, but indirectly, via some Web service (the server side). Looking at our class definition in Section 3, we can say that each independent client application is in the same "active class", defined implicitly. However, this is a class with restrictions, in which no pairing is necessary, and thus, no verification to join the distributed application is carried out. Moreover, when existing, the communication with the main device is unidirectional, in order to establish some kind of synchronization with the main audiovisual content. This communication usually uses proprietary protocols based on some ACR (Automatic Content Recognition) control.

Generally, companion screen applications must be installed in all devices prior to execution, making an obstacle to deployment. They are typically developed without any middleware or framework support, except those for ACR control generally third party proprietary solutions. The two main methods used are Audio Watermarking and Audio Fingerprinting, and which is most suited depends on a couple of factors.

Audio watermarking consists in analyzing the audio track to reveal positions in the signal where, considering the signal masking characteristics of the human audition, we can hide some digital codes without affecting the sound quality of the original. The main advantage of this technique is that, if you are a TV channel or a content owner, you can encrypt the data you are injecting in the stream, thus making other actors unable to exploit your tags. The data injection occurs on short intervals (like 2 seconds or less), so the technique provides good time accuracy. Drawbacks are that the bandwidth available is not so big (only several bytes in each audio segment) and that the technique is inhibited by long periods of silence where we can't hide anything. Usually there is enough space to hide a programme identifier and time information, or a trigger code, without it being noticeable or affecting the quality. As long as the application can pick up the watermark signal, it can tell you what you're watching and how far through it you are.

Audio fingerprinting consists first in taking an imprint (also referred as "signature") of the audio track of the video contents and store it into a database. The client (companion screen) device takes the same type of audio imprint on short timescale (like 5 to 10 seconds) and

sends it to the server which then searches for it into the stored imprints collection and returns the content ID when it's found. This technique has a main advantage: you don't have to be the owner of the contents to analyze it, so this is a perfect way to build synchronized services if you are not a TV channel or a cinema studio. But it has also many drawbacks: you must run a huge server infrastructure to compare the signatures, the signature doesn't allow to identify the distribution source, multi-lingual contents require several signatures, and finally you need to have access in advance to the fresh contents to generate signatures before their air time- live fingerprinting being a kind of challenge at high scale.

The great advantage of using ACR for synchronizing with the main audiovisual content is that the communication between the main device and the secondary devices does not depend on any protocol to be supported by the main device. In other words, the companion screen client applications are totally independent from the implementation of the main device, although can be dependent from the content owner, especially in the case of watermarking.

Some platforms for connected TV support companion applications. Yahoo Connected TV [11] includes a JavaScript API for integrating mobile devices. The platform uses the mDNSResponder library (part of the Bonjour platform) for service discovery and supports secure message exchange between the main device and the paired devices (individually, via SSL). Google TV[5] is a platform delivered by many connected TV manufacturers. The Google TV API allows smart phones and tablets to be used as enhanced remote controllers and also to display related content. The platform defines a pairing protocol (Google TV Pairing) and a message exchange protocol (Anymote). However, both Google TV and Yahoo TV platforms do not support transmission of media content between devices, do not have any specific functionality for media synchronization, and have no fault tolerant mechanisms associated to the distributed multimedia features.

Multi-device applications have recently received attention in terrestrial Digital TV systems. However, although some platforms provide middleware support to distributed multi-device applications, few applications are of this type. While based on open standards, the resulting systems usually are not open to anyone at all: only broadcasters and CE manufacturers have a say in what applications to run. With the advent of integrated broadcast/broadband systems the situation is changing little by little.

Ginga [1, 21] supports multi-device presentations for applications developed using the NCL language [36]. The middleware specification for ISDB-T DTV system delegates to commercial implementations how devices are paired. The current reference implementation of Ginga [38] uses a very simple service discovery process, and has support to only three classes (one passive class and two active classes). Dynamic specification of device classes is not supported. The implementation does not either offer mechanisms to identify devices registered in a class. Child (secondary) devices can only send event notifications to the parent device (e.g. main device) to maintain inter-media synchronization. Media content can only be sent from the parent to child devices. It is not possible to have Ginga accessing linked resources using technologies like UPnP (briefly explored in [8]).

ARIB STD-B23 Java-based middleware [2] defines an API to get resources from devices connected to a DTV receiver, providing basic device communication functionalities,

[5] http://www. google.com/tv

mechanisms for querying device status, and procedures for device service discovery. The platform offers limited support to UPnP, Bluetooth and USB device connections. The API does not address functionalities for media synchronization.

There are several proposals for integrating DVB/MHP middleware with the OSGi framework [25]. Their goal, to some extent, is to allow MHP Xlets to use resources associated to registered OSGi services. However, the approaches do not focus on multi-device presentations or on distributed media synchronization.

HbbTV middleware has already announced that a set of companion screen functionalities will be present in its next specification [13]. Such specification, which has no commercial or prototype implementation available to this date, aims at integrating interactivity into the TV by use of web technology, allowing for the use of companion screens (like smart phones and tablets) and more sophisticated synchronization scenarios.

In short, all main terrestrial DTV systems plan to offer better support for interoperable distributed multi-device applications, but today only Ginga has a fair support, but still needing improvements. Some research projects, however, address important issues on the definition of a framework to support interoperable multi-device applications.

Ginga reference implementation has adopted the architecture introduced in [37, 39], which has introduced the concept of class of devices and of hierarchical orchestration. Cesar et al. [9] proposed an architecture in which users share a common control environment and view the common content on the main screen. Secondary device discovery is done by an exchange of invitations to join the network using Bluetooth, WiFi, or IP Multimedia Subsystem (IMS). After devices have been discovered, they provide the descriptions of their physical, rendering, and interactive capabilities. Applications are described as a Software Oriented Architecture (SOA). Based on device and service descriptions, a matching algorithm together with basic recommendation facilities are used to determine where to render a media and how to provide user interaction in the most suitable way depending on the context of use. Each matched secondary device receives an initial SMIL 3.0 document [7] in which users may perform individual edits concurrently (to enrich the video watched on the main screen) and incrementally, creating a new SMIL document. The resulting micro-personal recommendation is the sent based on the social network information. All communication is done via SMIL documents. A Presentation module is responsible for parsing and rendering these documents on secondary devices.

The neXtream system prototype [26] (using Apple TV and iOS devices) integrates mobile applications, social networks, and video streaming using the Bonjour protocols for service discovery and delivery over IP networks. OSC messages enable mobile devices to be used as remote controllers in content navigation. MDCS (Multimedia Delivery and Control System) [23] dynamically generates multimedia applications, focusing on content adaption and on session mobility. The system generates SMIL code optimally configured to the characteristics of the integrated devices. Neither neXtream nor MDCS support media exchange between paired devices.

The proposal of an API aiming at providing multi-screen capabilities in traditional web applications with minimal effort is also explored by [5]. The API introduces two event types “DeviceConnected” and “DeviceDisconnected” which are triggered in case a secondary device is connected or disconnected. The notification mechanism abstracts from the mechanisms used to discover and pair the main device (TV) and secondary devices. After a secondary device is connected, the main application can send a request to it in order

to launch a specific web application. In case the user of the secondary device accepts the request, the companion web application will be launched and the success callback will be triggered in the main application by passing a message port (part of the HTML5 Web Messaging API) as input parameter. The main application can use the port.postMessage and port.onmessage to exchange data with the companion application. Just references are sent on the messages, no media objects nor streams. As proof of concept, the API was implemented as part of a prototype called FAMIUM[6].

Few efforts can be found outside the digital TV domain. Interactive Cinema has been explored by different systems, offering different types of user interaction. Most approaches are installations based on single-user or on very specific interaction mechanisms; only few of them use devices adjoined by viewers. AVIE [28] and LiveCinema [24] offer mechanisms (based on specific hardware and software) so that viewers may alter the pace and the content being projected.

DiS [14] and Active Presentation [6] are proposals for interactive slideshow presentations using multiple devices. DiS distributes the visualization of Microsoft Powerpoint presentations using IP multicast. It supports dynamic device registration and device hierarchical control. Active Presentation uses NCL to capture information comprising slideshow presentations, offering limited reproduction of the distributed content. Although these efforts provide a relevant set of functionalities, they only partially support some of the requirements discussed in previous section.

Distributed multi-device applications require developers to think about non-functional aspects not present when designing single-screen applications. This includes discovery, pairing, bidirectional multi-device group and unicast communication and synchronization, alignment of user preferences, etc. Looked at individually, none of these aspects are unsolvable today. Each aspect has some solution, usually proprietary, targeting specific device brands and specific domain of applications. However, there is no interoperable common baseline, no language support or software platform that offers high-level abstractions that hide or minimize the complexity of dealing with this distributed and heterogeneous execution environment, in all mentioned non-functional aspects, to aid developers that need to build multi-device applications. This is the main contribution of this Technical Report.

5. NCL SUPPORT TO MULTI-DEVICE APPLICATIONS

NCL 3.1 supports the four previously mentioned device classes defined by FSMDA: passive, active, media capture, and on demand media. In NCL, the <regionBase> element is associated with a device class where the presentation will take place. In order to identify the class (i) of devices, the *device* attribute of the element can receive the “class(i)” value.

The <regionBase> element also defines the capabilities a device aiming at registering to a class must have. The device class description, an RDF specification, is the content of a <metadata> element, child of the <regionBase> element that defines the class.

The device class description is based on CC/PP, using the UAProf framework [41]. The description model introduces the RDF DeviceClass (as shown in Table 2). It is composed by two simple attributes (maxDevices and minDevices) plus three instances of UAProf classes (which are subclasses of the generic element defined by CC/PP): SoftwarePlatform,

[6] http://www.fokus.fraunhofer.de/go/famium

NetworkCharacteristics and HardwarePlatform. The first two are extended with new attributes, as shown in Table 3 and explained ahead.

Table 2. *DeviceClass* definition.

DeviceClass	
Attribute	**Type**
fsmda:classType	String
fsmda:maxDevices	Number
fsmda:minDevices	Number
fsmda:HardwareRequirements	prf:HardwarePlatform
fsmda:SoftwareRequirements	prf:SoftwarePlatform
fsmda:NetworkRequirements	prf:NetworkCharacteristics

Table 3. Attributes incorporated to the UAProf classes.

SoftwarePlatform Class		
Attribute	**Type**	**Example**
fsmda:softwareParams	rdf:Bag	"KEY=12345"
NetworkCharacteristics Class		
Attribute	**Type**	**Example**
fsmda:pairingMethod	Literal	"UPnP", "QRCode", "none"
fsmda:supportedMessages	rdf:Bag	"HTTP", "RTP", "Watermark" "Fingerprinting"; "OCR"
fsmda:networkParams	rdf:Bag	"IP_ADDR=10.0.1.1"

FSMDA, and thus NCL 3.1, allows for limiting the number of devices (maximum and minimum) a class has. This mechanism comes together with a method to identify each individual device in a class. Constraining the number of registered devices in a class is important for applications that only make sense when used by a certain number of secondary devices. Identifying a device within a class is important to allow content personalization.

The *SoftwarePlatform* UAProf class has a new attribute, softwareParams, used to specify arbitrary parameters that might be used with the secondary devices' services (for instance, a key to be used by an application).

The *NetworkCharacteristics* UAProf class has three new attributes: *pairingMethod* stores the identification of the pairing mechanism that must be used by child devices (none by default); *supportedMessages* specifies the protocols and communication technologies that must be supported by child devices (for instance, HTTP messaging or audio watermarking for synchronization); and *networkParams* defines parameters for network services of the child devices (for instance, to restrict which address can be used by a secondary device).

The <regionBase> element can also define a set of regions on devices paired to its associated class. Regions define where media objects' content may be displayed, and are defined using <region> elements children of the <regionBase> element.

A media object is represented in NCL by <media> elements. These elements can have <property> child elements associating the display of the media object's content to a device

class and to a display region on devices paired to the class. Alternatively, a <media> element can refer (though its *descriptor* attribute) to a <descriptor> element that refers (through its *region* attribute) to a <region> element defined as a child of a <regionBase> element associated to some device class. Listing 1 illustrates both cases. In lines 26, 19, and 7, an image is defined to be displayed on region "left=25%, top=25%, width=100%, height=100%" of all devices paired to class (5). In lines 27 to 34, another media content is defined to be displayed in the same region of all devices paired to class (5). If both contents are displayed simultaneously, the second one (line 27 to 34) superposes the first one because its *zIndex* attribute has a higher value.

```
1.  <ncl id="example" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
2.    <head>
3.       <regionBase >
4.          <region id="loc01" width="100%" height="100%" zIndex="1"/>
5.       </regionBase>
6.       <regionBase device="class(5)">
7.          <region id="loc02" left="25%" top="25%" width="50%" height="50%" zIndex="2"/>
8.          <metadata>
9.                RDF Tree
10.         </metadata>
11.      </regionBase>
12.      <regionBase device="class(7)">
13.         <metadata>
14.               RDF Tree
15.         </metadata>
16.      </regionBase>
17.      <descriptorBase>
18.         <descriptor id="desc01" region="loc01"/>
19.         <descriptor id="desc02" region="loc02"/>
20.      </descriptorBase>
21.…
22.  </head>
23.  <body>
24.…
25.      <media id="example01" src="../media/video.mp4" descriptor="desc01"/>
26.      <media id="example02" src="../media/image.png" descriptor="desc02"/>
27.      <media id="example03" src="child-device://class(7).device(1)/VideoCapture">
28.         <property name="device" value="class(5)"/>
29.         <property name="left" value="25%"/>
30.         <property name="top" value="25%"/>
31.         <property name="width" value="50%"/>
32.         <property name="height" value="50%"/>
33.         <property name="zIndex" value="3"/>
34.      </media>
35.      <media id="example04" src="../app/advert.ncl" type=" application/x-ncl-NCL"
                                                              descriptor="desc02"/>
36. . .
37.  </body>
38.  </ncl>
```

Listing 1. Examples of class definition and media objects association to classes.

In NCL, the location of a media object's content is defined in the *src* attribute of the corresponding <media> element. FSMDA has extended the previous version of NCL adding a new URL schema (*child-device*), so that content captured by secondary devices paired to a Media Capture Class could be referred. The complete URL is:

child-device://<class_id.device_id>/<resource_id>

in which, *device_id* is the device identifier and *resource_id* identifies the resource associated to the media object. For example, the URL

child-device://class(3).device(1)/AudioCapture

is used to access an audio capture service (identified by the "AudioCapture" string) from a child device with index '1' (the first paired device) in a class with index '3'. In Listing 1, lines 27 and 28 specify that the video coming from the first paired device to "class(7) is displayed on all secondary devices paired to "class(5).

In NCL, a special <media> element of "application/x-ncl-settings" type keeps global variables that can be manipulated by NCL applications. Some of these variables are environment variables whose values are defined by the NCL Player. Some other variables are defined by the application authors. Table 4 lists the new global variables added to the NCL 3.1 specification whose values are attributed by the NCL Player. Some of them come from the class descriptions and some from NCL-Player's internal mechanisms (e.g., class(i).devNumber, which is updated every time a child device joins or leaves a class).

Table 4. FSMDA's global variables.

class(i).type	"passive" | "active" | "mediaCapture" | " onDemandMedia"
class(i).screenSize	Screen size dimension
class(i).graphicPlaneSize	Resolution set for the screen graphics plane
class(i).audioType	" mono" | " stereo" | "5.1"
class(i).devNumber	Number of registered devices
class(i).maxDevices	Maximum number of devices
class(i).minDevices	Minimum number of devices
class(i).info	List of media players that paired devices must support
class(i).softwareParams	List of software parameters associated to that class
class(i).networkParams	List of network parameters associated to that class

Other variables (states and arbitrary parameters) can also be dynamically associated to classes. Secondary device registrations and deregistration on these classes can be conditioned on these dynamic parameters and states. Application's modules running on (parent) devices are able to capture information from device classes they control (device that are paired to them) using the following format:

- *class(i).var*, where *i* is the index associated to a device class, and *var* is the name of an arbitrary variable;
- *class(i).device(j).var*, where *i* is the index associated to a device class, *j* is the index associated to a paired device (following the order of pairing), and *var* is the name of an

arbitrary variable modifiable by the paired device (which means that some local variables of paired devices can be accessed by the parent device).

Paired devices are able to read their indexes of registration on active classes through the *child* namespace and its *child.index* variable (this can be used, for example, to personalize the content being presented on each device).

Figure 15 illustrates an example of variable manipulation. In (1) two devices pair themselves with a TV device. The TV acts as a parent device and associates indexes to its paired (child) devices; indexes that they are able to read through the child.device variables. In (2) a paired device modifies the value of a local variable (var). As a consequence, in (3) the variable change is reflected on a variable accessible to the parent device.

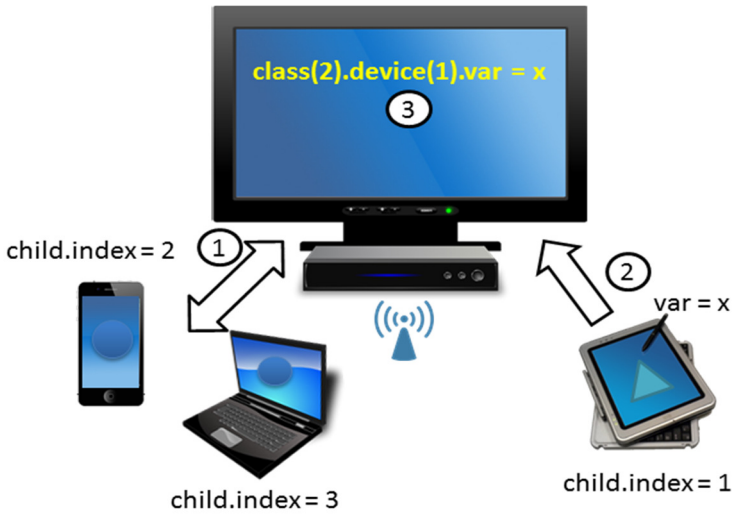


Figure 15 – Variable manipulation

NCL applications can embed other NCL applications. In Listing 1, line 36, an advert application is embedded and defined to run on devices paired to "class(5)". If class (5) is an active class, the application will be processed by secondary child devices paired to the class. If in the advert application other classes are defined, the paired child secondary device running the application becomes a parent device for these classes, to which other secondary child devices can pair to, defining a hierarchical orchestration tree. The single constraint imposed by NCL is that the orchestration process indeed induces a tree; it is not allowed closed path in the defined graph, i.e., a device cannot be a descendant of itself.

NCL defines some default classes, for simplicity. The "class(1)" is a passive type class. The "class (2)" and "class (3)" are classes of active type. Devices registered in "class (2)" must be able to present any media object type specified by a specific NCL Player implementation, including imperative NCLua, declarative NCL and HTML players. Devices registered in "class (3)" must be able to present any media object type specified by a specific HTML Player implementation, including declarative NCL (if it is supported) and HTML players. Moreover, the main device is considered to be paired to a private active class (in which no other device can pair to) that is declared by default. In Listing 1, lines 3 to 5 define the default class for the main device in which the whole screen (line 4) is reserved for the main video display (line25).

6. GINGA SUPPORT TO MULTI-DEVICE PRESENTATIONS

FSMDA is divided in two subsystems: Parent Device and Child Device. The main device always has the Parent Device subsystem. Secondary device always have the Child Device subsystem, but can also have the Parent Device subsystem, if it is not a leaf in the orchestration tree. The components of the subsystems establish their relationships and dependencies through the definition of programming interfaces (API).

The Parent Device subsystem is responsible for orchestrating the necessary (local and remote) resources, according to the logic of the distributed multi-device application in execution. It must be in agreement with all requirements stated in previous section, and must provide support to:

- define device classes;
- dynamic associate secondary devices to device classes it supports, using a generic registration method (*hot spot* of the architecture) and only if these devices fulfill the pre-requirements for association;
- use different network services and communication protocols (*hot spots* of the architecture) for the registration and the orchestration of secondary devices.

The following components compound the Parent Device subsystem.

The "Parent Device Pairing Manager" component manages the pairing features associated to all specified device classes. The component uses the host OS's network modules to offer pairing mechanisms to register secondary devices to each device class. This component uses the API in Tables 5. 6 and 7.

The "Parent Device Communication Manager" component handles commands associated to media content presentation (start, stop, pause, etc.), and to set values to properties of media objects associated to some device class. Presentation commands are processed by the component and delivered to associated secondary devices (running Child Device subsystem). The Parent Device Integration Manager is also the liaison to receive content and notification messages coming from secondary devices. To control which devices are paired and to which devices orchestration commands and notifications are sent to and from the component uses the services of the Parent Device Pairing Manager component. This component uses the API in Table 8. There must be one Parent Device Pairing Manager component for each media object to be presented on the secondary devices.

The Child Device subsystem is deliberately very simple. It aims at housing software modules present on devices compatible with third-party platforms. The following components compound the Child Device subsystem.

The Child Device Pairing Manager is responsible for controlling the pairing activities, which involves communicating with the Parent Device Pairing Manager. The component includes the pairing service subcomponents for each device class a Child Device subsystem is able to register. All communication is done using components included in the secondary device's network service. This component uses the API in Table 5, and 6.

The Child Device Communication Manager receives commands from the parent device to each device class the Child Device subsystem is able to pair. The component can also send notification messages and content to the associated parent device. Its subcomponents include those provided by the secondary device's network service package to communicate with the parent device. This component uses the API in Table 8. There must be one Child

Device Pairing Manager component for each media object to be presented on the paired secondary device.

6.1 FSMDA APIs

Table 5 presents the API for class handling, as incorporated by the Ginga reference implementation. This is the API between the NCL Player running on the parent device and the Parent Device Pairing Manager component.

Table 5. API for class handling.

Implemented by:	Operation (input parameters)
Parent Device Pairing Manager	**addDeviceClass** (integer index)
Parent Device Pairing Manager	**removeDeviceClass** (integer index)
Parent Device Pairing Manager	**addDeviceClassDescription** (string classType, integer maxDevices, integer minDevices, string HardwareReq, string SoftwareReq, string NetworkReq)

The API for discovery and pairing mechanisms depends on the network platform used and is presented in Table 6. This is the API between the Parent Device Pairing Manager and each Child Device Pairing Manager. Note that the Child Device index of registration (an integer value) is the output of the *joinClass* operation.

Table 6. API for service discovery and device pairing.

Implemented by:	Operation (input parameters)
Parent Device Pairing Manager	**deviceDiscovery (**integer classIndex**)**
Parent Device Pairing Manager	**addDeviceToClass (**integer classIndex, string deviceAddr, string deviceDesc, integer deviceIndex**)**
Child Device Pairing Manager	**joinClass (**integer classIndex, string classDesc**)**

Since NCL Players maintain the global variables in their Settings node, an API must be offered to the Parent Device Pairing Manager to set and get values of these variables. Table 7 shows this API between the Parent Device Pairing Manager and the NCL Player.

Table 7. API for handling classes' variables.

Implemented by:	Operation (input parameters)
NCL Player	**getDeviceClassVariableValue** (string name, string value)
NCL Player	**setDeviceClassVariableValue** (string name, string value)

NCL Player communicates with media players (local or remote) through a Media Player API. A Parent Device Communication Manager component is the mediator of a remote media player in the parent device. Indeed, the single mediator of all media players used by a same device class. Similarly, a Child Device Communication Manager component is the mediator of a remote NCL Player that controls a media object presentation on the child device. Every message sent to a Parent Device Communication Manager from an NCL

Player is passed to all corresponding Child Device Communication Manager components, using some (unicast or multicast) network protocol. Each Child Device Communication Manager component then sends the message to the corresponding media player. Likewise, every message sent to Child Device Communication Manager from a media player is passed to the corresponding Parent Device Communication Manager component using some network protocol (unicast communication), which then sends the message to the corresponding NCL Player.

In order to put a media object in execution in any child device, the NCL Player in the parent device must firstly instantiate the appropriate Parent Device Communication Manager (one for each media object to play), which will be in charge of instantiate the corresponding Child Device Communication Managers in every paired child device (one for each media object in each paired device), which will be in charge of instantiate the appropriate media player (one for each media object in each paired device) in the child secondary that will present the media object's content. Note thus that each Parent Device Communication Manager can be bound to several Child Device Communication Managers. More precisely, to a number of Child Device Communication Managers equal to the number of paired devices to a class to which the media object will be delivered for presentation. Messages coming from Parent Device Communication Manager to Child Device Communication Managers can use unicast or multicast communication. The same message received by the Parent Device Communication Manager from the NCL Player is sent to all paired devices. Messages coming from Child Device Communication Managers to the Parent Device Communication Manager always use unicast communication. Parent Device Communication Manager notifies the NCL Player in the parent device of each notification received from each corresponding Child Device Communication Manager. The NCL Player is not able to know from which paired secondary device the notification comes; the NCL Player only communicates to a class of devices.

Between each pair of entities A/B (NCL-Player/Parent-Device-Communication-Manager; Parent-Device-Communication-Manager/Chid-Device-Communication-Manager; and Child-Device-Communication-Manager/media-player) there is a similar API, illustrated in Table 8.

Table 8. Media Player API.

Implemented by:	Operation (input parameters)
B	**prepare** (mediaType media)
B	**addEvent** (eventType event)
B	**postAction** (string actionType, ID eventId)
A	**notifyEventTransition** (ID eventId, transitionType transition)
B	**requestPropertyValue**(string name)
A	**notifyPropertyValue** (string name, string value)
B	**setPropertyValue**(string name, string value, string duration, string by)
A	**notifyError**(string message)

Entity A knows the corresponding entity B instance from the moment it is created and from then on it begins to communicate directly with this instance. Similarly, entity B knows the

corresponding entity A, for example through a registration process executed soon after the entity B instantiation.

The *prepare* operation, issued by the NCL Player, shall inform the following parameters to the secondary device's media players: the properties associated with the media object to which the media player has been created, the list of events (presentation, selection, attribution, etc.) that need to be monitored by the media players, and the location of the media content to be executed/presented.

Events that need to be monitored are identified by the *eventId* parameter. The events derived from the *whole content anchor* and *main content anchor* shall also be identified.

If the content cannot be located, or if the media player does not know how to handle the content type, the media player shall finish the *prepare* operation without performing any action and report an error message.

Events can be added or removed from the list of events using the *addEvent* and *removeEvent* operations, respectively.

The *setPropertyValue* interface allows the NCL Player to set values to properties of the media object in execution controlled by the secondary device's media player. For example, using the *setPropertyValue* the host language player can pass an input parameter used by the media players in running the presentation. When setting a new value to a property the change is instantaneous by default (parameter *duration*="0"), but the change may also be carried out during an explicitly declared duration, specified by the *duration* parameter. In this last case, the change from the old value to the new one may be linear by default (parameter *by*="indefinite"), or carried out step by step, with the pace specified by the *by* parameter.

The *requestPropertyValue* interface allows the NCL Player to get property values of the media-object in execution controlled by the media players. The value is returned when the media player notifies the NCL Player by means of the *notifyPropertyValue* interface.

NCL Player starts, stops, pauses, resumes, aborts, etc. media player presentations via *postAction* interface, specifying the desired action (start, stop, pause, resume, abort, etc) to be executed on the event state machines.

Media players must notify the host language player of changes in the event state machines it controls. Event state changes are notified via *notifyEventTransition* interface.

Since a parent device can retrieve media content from child devices paired to Media Capture class it defines, the API presented in Table 9 is established between each pair of entities A/B (NCL-Player/Parent-Device-Communication-Manager; Parent-Device-Communication-Manager/Chid-Device-Communication-Manager; and Child-Device-Communication-Manager/media-player) to accomplish this task.

Table 9. Media Player API.

Implemented by:	Operation (input parameters)
B	**requestContent** (integer deviceIndex)
A	**sendContent** (char* content)

Besides the modules that comprise FSMDA, child secondary devices must have modules to execute their main tasks in processing distributed multi-device applications. For example,

components do display content receive by a device paired to a passive class, media players to process media objects received by a device paired to an active class, components to capture content to be sent from devices paired to a media capture class, etc.

6.2 Input Device Control Model

In the beginning of an NCL document presentation, all input units associated with devices of a multi-device application are under control of the main device.

In NCL, some media players may gain control of the input devices that previously were controlled by the NCL Player. For example, when a <media> element displayed on a secondary device paired to an active class is in focus and the ENTER key is pressed, the secondary device in charge of the presentation gains control of all its input units and all input units of devices that are in classes that will be its descendants (classes for which it will be the parent device). The media player can then follow its own navigational rules. When the "BACK" key is pressed, the control of all previously mentioned input units is returned to the parent device. As usual in NCL, the focus will go to the element identified by the *service.currentFocus* attribute of the Settings node (<media type="application/x-ncl-settings).

The control passing is notified to the media player via the *notifyInputControl* interface, as shown in Table 10.

Table 10. Input Control API[7].

Implemented by:	Operation (input parameters)
Media player	**notifyInputControl** ()
Input controller	**requestInputControl** (deviceType device, keyListType keyList)
Media player	**notifyInput** (deviceType device, sensorType sensor)
NCL Player	**nestInput** (unsignedInteger nestingLevel)

Upon receiving the notification, the media player must register which device types it wants to control (by default all keyboards, motion sensors, and remote controls) and which particular list of keys (in the case of keyboards and remote controls). This is done via the *requestInputControl* interface. From then on, each registered input is passed to the media player via the *notifyInput* interface: a key identification or the sensor position.

As the input control model is recurrent, a media player can also pass the control acquired to an internal entity, and so on. However, each time a descendant entity gains control, the media player shall notify the host language system via the *nestInput* interface. When a BACK key is pressed, control must be passed back to the parent entity, until reach the root language player that can now pass control to another media player.

It should be noted that the hierarchical input control may refer to any input device spread in a distributed environment (in a multiple device execution).

6.3 Fault recovery mechanisms

Platforms for distributed hypermedia applications are potentially prone to faults, but some of them can be handled and recovered. In FSMDA, it is important that the Parent Device

[7] The used data types are defined in Annex II

Communication Manager component is equipped with mechanisms for fault recovering, since it centralizes all application data and execution status shared with child Device subsystems.

If the Parent Device Communication Manager detects that any device class is empty, or if an active class has fewer devices than its minimum limit, it issues a notification, to warn the NCL Player that relationships involving media objects associated to that class should be longer considered during the presentation. Ongoing presentations of media objects associated to the class should be aborted.

To avoid synchronization mismatches during the application presentation, it is important to periodically verify the execution progress of media objects handled by devices registered in active classes. Upon detecting an invalid state (e.g., when a secondary device pairs for a second time, after temporarily losing connectivity), the Parent Device Communication Manager component issues a notification to the NCL Player allowing that the faulty secondary device presentation is resumed at a valid point in time.

7. SYSTEM TEST CASES

An NCL Digital TV application, with a 12 minute video piece about the FIFA World Cup 2014, was developed aiming at testing all requirements related to the orchestration of *active* and *UPnP* classes, as well as the fault tolerant mechanisms. Another NCL Digital Cinema application tested the requirements related to the orchestration of *active*, *passive* and *media capture* classes. Still another NCL Slideshow presentation was developed to test the requirements related to the orchestration of *active* and *passive* classes, and the hierarchical orchestration model.

In the new Ginga reference implementation, Parent Device subsystem has been developed in C++. The implementation supports different media backend (such as DFB, SDL, etc.) and was ported for Linux and Windows. The extended Ginga reference implementation allows for orchestrating active, passive, media capture and UPnP classes. The previous Ginga reference implementation already offered fault detection and recovery mechanisms, making the middleware resilient to several kinds of faults [32]. The current implementation has been extended with the features introduced in Section 6.2. In case of faults or invalid states, the Parent Device Communication Manager notifies the Recovery module of Ginga, which then notifies the Presentation System module, in order to guarantee a consistent presentation.

Child Device subsystems have been developed in C++ for Linux and Windows platforms, supporting services for active and passive classes. In these implementations, active classes can be defined for any combination of media players present in the Ginga specification for ISDB-T (including monomedia players, Lua engine, HTML players and NCL players) [1]. Other Child Device subsystems were developed for the Android platform to include services for active, passive and media capture classes.

7.1 Digital TV Scenario

In the main device a broadcasting channel is tuned to receive the application (its specification and content – the main video about the most beautiful goals of all World Cups and other additional media objects).

When the main application starts, secondary devices can pair to *active* class services and *UPnP* class services. At a certain moment, viewers are able to navigate as a group on a

menu of World Champions' country flags presented in the TV screen. If a flag is selected by a viewer, the scores of the final matches when the country was champion are displayed in the TV screen. At the same time, paired devices, registered in the *active class,* start presenting an invitation (companion application) to move all media objects being presented on the TV set, except the main video, to the secondary devices' screens. If the invitation is accepted, individual navigation on the menu of champions, in each secondary device, is then possible. Figure 16 presents two snapshots illustrating the situation: the first one shows the main screen while presenting the World Champions information; the second one shows the result of moving the World Champions information to the secondary devices, with appropriate adaptations.

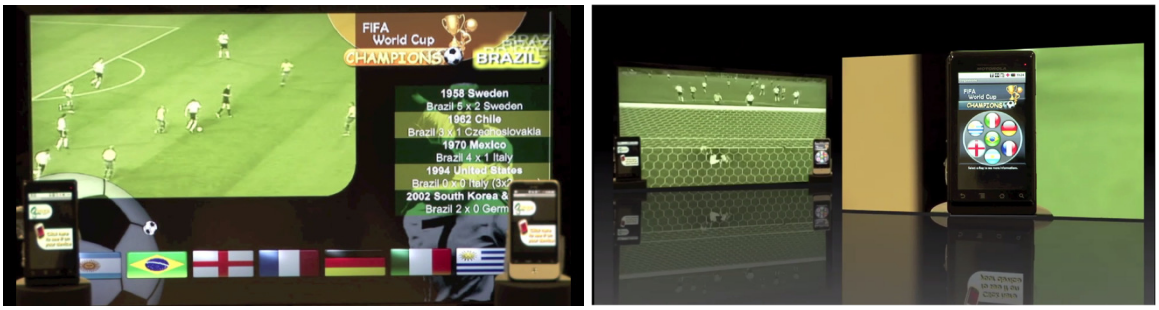


Figure 16 – Moving information from the main screen to secondary devices

If a device registers itself in the *active* class after all media objects are brought to other secondary devices, and if no fault handling mechanisms was present, the late device would not present any interactive feature. Using Ginga, however, the main application is able to detect the inconsistent state and to trigger the recovery mechanism to present the menu of champions on the late device's screen. If during the presentation of additional information on the secondary devices, these devices were turned off, the main application can detect that the *active* class has become empty and trigger the display of the additional information back to the TV screen.

The multi-device application can also display videos about the best moments of the 2002 World Cup final match. The videos may be retrieved and started by the main application. When a video starts, it becomes visible to all registered UPnP compatible secondary devices registered to the *UPnP* class. When the main application ends or it stops the video object associated to the *UPnP* class, the video is removed from the shared list and ends its presentation on the UPnP secondary devices. Figure 17 shows a snapshot of this moment.

Figure 18 – Best moments of the 2002 World Cup final match

The application continues until the point an advert about the 2014 World Cup is sent to secondary devices paired to the active class, inviting their viewers to buy tickets for the games of the 2014 World Cup. Each viewer can navigate on the received content independently, to buy the tickets, as shown in Figure 18.

Figure 18 – Buying tickets for FIFA 2014 World Cup

Near to the end of the main video presentation, the main application starts a game, which is sent to the two first registered devices of the *active* class (the "penalty shooter" companion application to the first device and the "goalkeeper" companion application to the second one). The companion applications prompt their viewers for choosing a direction. Afterwards, an animation displays the result of the penalty shootout on the TV screen: if both, goalkeeper and shooter, choose the same side, the result is a save; if the opposite happens, the shooter scores a goal. Figure 19 illustrates this last game.

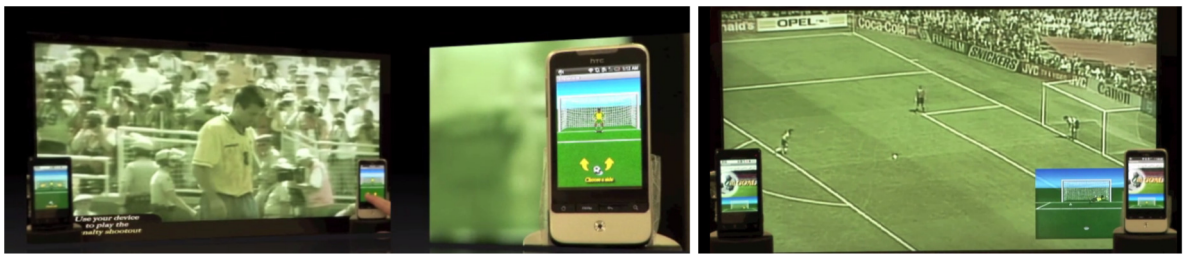


Figure 19 – Gaming between secondary devices

We have also injected a fault during the penalty shootout game. If after starting the game, all devices in the *active* class were turned off but one (the first to join), the fault is detected and the viewer is notified that the game was not executed due to lack of opponents.

7.2 Digital Cinema Scenario

In this test case, the main application runs on a computer plugged to a projector playing an interactive story in which viewers are asked about decisions the protagonist must take.

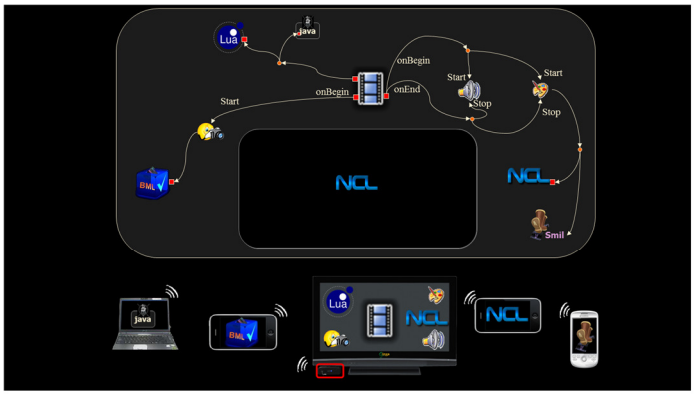
The main application starts offering pairing to the *active*, *passive* and *media-capture* classes. At a decision moment, viewers registered to the *active* class receive an object with a voting demand. The votes are computed by the main application. After the decision period, the voting demand presented on the companion devices stops and the main video resumes according to the choices made by the majority. Devices paired to the passive class

are then able to view, in a low quality video stream, excerpts from the scenes of the options which were not chosen.

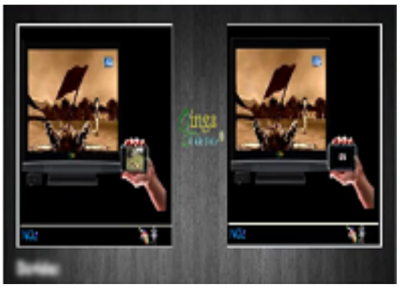
The *media-capture* class is explored at the end of the movie presentation, when an advertisement is presented. The main application synchronizes the content being exhibited on the movie projector with an audio being captured by the first device paired to the *media-capture* device class. When in the advertisement a character makes a phone call, a companion application emulates a phone call, simultaneously. The application asks the viewer whether s/he wants to answer the call saying aloud where s/he is seated, in order to win a prize. If the viewer chooses to "answers" the call, the application starts capturing the audio coming from the secondary device and sends it to the main application that reproduces it to the whole audience.

7.3 SlideShow Presentation Scenario

In this test case, the main application runs a slideshow presentation, and offers pairing to a single device in an *active* class. The presentation speaker pairs its device (a tablet) with this class. It then receives a companion (NCL) application that allows for controlling which slide will be displayed. The application running on the speaker's tablet in its turn offers pairing to a *passive* class, and associates media objects to this class. Attendees with devices paired to this passive class receive a pre-rendered stream from the tablet, synchronized with each slide presentation.



Main Screen



Speaker's companion screen (next slides)

Attendee's companion screen (additional information)

Figure 20 – Talk about multi-device applications

8. CONCLUSIONS

The specification of the software architecture to support distributed multi-device applications based on a hierarchical orchestration model is the main contribution of this work. Instantiated as part of the new version of Ginga, this architecture is operational today, and is under evaluation to integrate the ITU-T H.761 Recommendation. Indeed, the changes

proposed to the NCL language have already been approved and added to the ITU-T NCL 3.1 version.

Some issues concerning the architecture still deserve research attention and are in our future work plans. Inter-media synchronization is one of them, in all its extents. To be more specific, inter-media synchronization refers to the preservation of the temporal dependences between different media objects, with the same or different content (e.g., lip-sync). A particular sub-type of inter-media synchronization is referred to as multi-source (or inter-sender) synchronization, which aims to synchronize the playout of several media content originated from different senders (or sources). As an example, assume different content about a life event being transmitted by many users of secondary devices paired to a Media Capture class. Another sub-type of inter-media synchronization is referred to as inter-device synchronization, in which different media contents can be played out in separate devices synchronously. This is exemplified by almost all applications mentioned in this paper, in which content presented by the main application is synchronized with content presented on secondary devices. Finally, we are also interested in the simultaneous synchronization of the playout of a specific media stream in various devices. This is usually known as inter-destination media synchronization [30]. As example, assume a content broadcasted from the main application to secondary devices paired to the passive class. In all these inter-media synchronization cases, the devices can either be physically close-by or far apart. In the current version of FSMDA we only have coarse synchronization: they may have a "slight" synchronization mismatch; and this "slight" can be unbearable when devices are far apart.

Still regarding the architecture, we must evaluate if the hierarchical model used can be relaxed to allow secondary devices paired to a same class to communicate without the mediation of the parent device. This can be very interesting since media capture by some secondary devices can address other secondary device to present the content. A direct communication can save network bandwidth. Listing 1 presents an example, in which the video coming from the first paired device to "class(7) is displayed on all secondary devices paired to "class(5) (lines 27 and 28). Moreover, since usually secondary devices paired to some active class are in the same network, when the parent device command them to play a media object passing the URL location of its content, maybe it is better that just one child device retrieves the content and then distributes it to all other child devices, if the content server is in another different network; again saving bandwidth. In addition, how devices communicate is another important issue. Identification of more complex network architectures, in which devices can be connected through P2P/mesh networks is in our future work plan.

Our future work list still includes:

- Design of better declarative abstractions to support the functionalities provided by the proposed architecture (in this direction, NCL 4.0 is already under discussion in ITU-T H.761 Working Group). Just as an example, if an NCL application gets a property value from a media object addressed to an active device class, it can receive different values from the paired devices. What is the value to be used?
- Revision of the considered ontology and metadata used to describe the device's services.
- Revision of the Input Control API to allow multi-touch technology, which can reflect in the NCL support to multiple focus point.

- A secure communication support, as well as a better authentication and content certification mechanisms, necessary for protected media and other applications with relevant security requirements.
- Instantiation of FSMDA using similar platforms other than UPnP (e.g. OSGi) to verify the completeness of the offered APIs.
- A conformance test suite to be applied to devices, as part of the pairing procedure, before allowing them to join the platform.
- Design of system test cases exploring groundbreaking multi-device applications different from those following web-based model, in order to raise possible new requirements for the architecture.
- Search for other collective space domains in which FSMDA can be applied, as for example health applications.

ACKNOWLEDGMENTS

The authors would like to thank TeleMidia Lab's researchers who provided a thoughtful discussion of this work. As well authors gratefully acknowledge the grant from CNPq, CAPES and MCT.

REFERENCES

1. ABNT. *Ginga-NCL for Fixed and Mobile Receivers—XML Application Language for Application Coding.* ABNT NBR 15606-2, 2nd ed., Brazilian Nat'l Standards Organization, 2011.
2. ARIB – Association of Radio Industries and Businesses. *Application Execution Engine Platform for Digital Broadcasting*. ARIB STD-B23 Version 1.2, 2004.
3. Audible Magic Corporation. Digital Fingerprinting & Video Content Recognition Enabling New Forms of Interactive Advertising. White Paper. June 7, 2011. http://audiblemagic.com/white-papers/Digital_Fingerprinting_Enables_New_Forms_of_Interactive_Advertising.pdf
4. Barkhuus, L., Brown, B. Unpacking the television: User practices around a changing technology. *ACM Transactions on Computer-Human Interaction* (TOCHI) 16, 3 (2009).
5. Bassbouss, L., Tritschler, M., Steglich, S., Tanaka, K., Miyazaki, Y. Towards a multi-screen application model for the web. In *Proc. of IEEE 37th annual computer software and applications conference workshops*, COMPSACW 2013, July 2013.
6. Brandão, R., França, P., Medeiros, A., Portella, F., Cerqueira, R. The CAS Project: A general infrastructure for pervasive Capture and Access systems. In *Proc. of the 28th Annual ACM Symposium on Applied Computing*. (2013), 975-980.
7. Bulterman, D. C.A., Rutledge, L. W. *SMIL 3.0 - Flexible Multimedia for Web, Mobile Devices and Daisy*. Talking Books. 2nd ed. Springer, 2009. ISBN: 978-3-540-78546-0.
8. Cattelan, R. G. et al. Watch-and-Comment as a Paradigm toward Ubiquitous Interactive Video Editing. *ACM Transactions on Multimedia Computing, Communications and Applications*, v. 4 (4), No. 28, 2008.
9. Cesar, P., Bulterman, D., Jansen, A., Geerts, D., Knoche, H., Seager, W. Fragment, Tag, Enrich, and Send: Enhancing Social Sharing of Video. *ACM Transactions on Multimedia Computing, Communications and Applications.* Vol 5. No. 3. August 2009. http://doi.acm.org/10.1145/1556134.1556136

10. Civolution. Automated Content Recognition: creating content aware ecosystems. White Paper. *CSI Magazine*. September 2012. http://www.csimagazine.com/csi/whitepapers/ACR%20Creating%20%20content-aware%20ecosystems%20-Civolution%20White%20Paper%20-%20Sept%202012.pdf
11. Cortez, J., Shamma, D. A., Cai, L. Device Communication: A Multi-Modal Communication Platform for Internet Connected Televisions. In *Proc. of the 10th European conference on Interactive TV and video*, ACM Press (2012), 19-26.
12. ETSI. *Digital Video Broadcasting (DVB). Multimedia Home Platform (MHP) Specification 1.1.1*, ETSI TS 102 812, ETSI Standard, 2003.
13. ETSI. *Hybrid Broadcast Broadband TV*. ETSI TS 102 796 V1.2.1 (November, 2012) http://www.etsi.org/deliver/etsi_ts/102700_102799/102796/01.02.01_60/ts_102796v010201p.pdf
14. Gaspar, B. et al. A New Approach for Slideshow Presentation at Working Meetings. In *Proc. of ConfTele* 2007, Peniche, Portugal, 2007.
15. Geerts, D., Cesar, P., and Bulterman, D. The implications of program genres for the design of social television systems. In *Proc.of the 1st international conference on Designing interactive user experiences for TV and video*, (2008), 71-80.
16. GfK MRI. The GfK MRI iPanek Reporter. Tablets and Multi-Tasking. Fall 2012.
17. Google.The New Multiscreen World - Understanding Cross-platform Consumer Behavior. August 2012. http://www.slideshare.net/smobile/the-new-multiscreen-world-by-google-14128722
18. Hess, J., Ley, B., Ogonowski C., Wan, L. and Wulf, V. *Jumping between Devices and Services: Towards an Integrated Concept for Social TV.* University of Siegen, Siegen, Germany. July, 2011.
19. Huang, E. M., Harboe, G., Tullio, J., Novak, A., Massey, N., Metcalf, C. J., Romano, G. Of social television comes home: a field study of communication choices and practices in tv-based text and voice chat. In *Proc. of the 27th international conference on Human factors in computing systems*, (2009), 585-594.
20. ITU-R. *Integrated broadcast-broadband systems*. . Report ITU-R BT.2267. BT Series. Broadcasting service. May, 2013.
21. ITU-T. *Nested Context Language (NCL) and Ginga-NCL for IPTV Services*. ITU-T Recommendation H.761, 2009. Geneve, Switzerland. June, 2011.
22. Kang, D. O., Kang, K., Choi, S., Lee, J. UPnP AV architectural multimedia system with a home gateway powered by the OSGi platform. *IEEE Transactions on Consumer Electronics*, 51(1), 87-93. 2005.
23. Kernchen, R. et al. Intelligent multimedia presentation in ubiquitous multidevice scenarios. *IEEE Multimedia*, 17(2), 52-63. 2010.
24. Lew, M. Live cinema: an instrument for cinema editing as a live performance. In *Proc. of ACM SIGGRAPH* (2004).
25. Lin, C. L. et al. Classification and Evaluation of Middleware Collaboration Architectures for Converging MHP and OSGi in a Smart Home. *Journal on Information Science and Engineering*, 25(1): 1337–1356, 2009.
26. Martin, R.et al. neXtream: A Multi-Device, Social Approach to Video Content Consumption. In *Proc. of the IEEE Consumer Communications and Networking Conference* (CCNC'10) (2010).

27. Matsumura, K., Kanatsugu, Y. HybridCast System and Technology Overview. *Broadcast T*echnology No.43, Winter 2011.
28. Mcginity, M., et al. AVIE: a versatile multi-user stereo 360 interactive VR theatre. In *Proc. of workshop on Emerging displays technologies: images and beyond: the future of displays and interaction*. ACM (2007).
29. MEDIATVCOM. Decrypting the Second Screen Market. November, 2012. http://blog.mediatvcom.com/wp-content/uploads/2012/12/White-Paper-Second-Screen-mediatvcom.pdf
30. Miso. The Miso Sync Experiment: Delivering real-time, relevant information on the second screen. May, 2011. http://miso.io/mU0zJL
31. Montagud, M., Boronat, F., Stokking, H., van Brandenburg, R. Inter-destination multimedia synchronization: schemes, use cases and standardization. *Multimedia Systems Journal*, 18(6), 459-482, November 2012.
32. Moreno, M.F. et al. Resilient Hypermedia Presentations. In *Proc. of XXI IEEE International Symposium on Software Reability Engineering (Workshop W2, Software Aging and Rejuvenation)*. San Jose, USA. (2010).
33. Nielsen. Double Vision – Global Trends in Tablet and Smartphone Use While Wartching TV. May 2012. http://www.nielsen.com/us/en/newswire/2012/double-vision-global-trends-in-tablet-and-smartphone-use-while-watching-tv.html
34. Roy, C.S., Galarneau, B. The Second Screen and Television. *Evolumedia White Paper Series*: Overview and growth perspectives (available at http://www.cmf-fmc.ca/documents/files/about/publications/Report-1-Second-Screen.pdf, October, 2012); Benefits & Impacts (available at http://www.cmf-fmc.ca/documents/files/about/publications/Second-Screen-and-TV-Report2.pdf, March, 2013); Production and Deployment (available at http://www.cmf-fmc.ca/uploads/reports/29-second-screen-and-tv-report3.pdf, August, 2013).
35. Soares L.F.G., Rodrigues R.F. *Nested Context Model 3.0 Part 1 – NCM Core*. MCC 18/05 Technical Report. Informatics Department of PUC-Rio. Rio de Janeiro. 2005. ISSN 0103-9741. Available at: http://www.telemidia.puc-rio.br/sites/telemidia.puc-rio.br/files/Part1 NCM 3.0.pdf.
36. Soares L.F.G., Rodrigues R.F. *Nested Context Language 3.0 Part 8 – NCL Digital TV Profiles*. MCC 35/06 Technical Report. Informatics Department of PUC-Rio. Rio de Janeiro. October, 2006. ISSN 0103-9741. Available at: http://www.ncl.org.br/documentos/NCL3.0-DTV.pdf.
37. Soares L.F.G. *Nested Context Language 3.0 Part 12 – Support to Multiple Exhibition Devices*. MCC 03/09 Technical Report. Informatics Department of PUC-Rio. Rio de Janeiro. January, 2009. ISSN 0103-9741 Available at: http://www.telemidia.puc-rio.br/sites/telemidia.puc-rio.br/files/Part 12 - NCL3.0-MD.pdf.
38. Soares, L.F.G. et al. Ginga-NCL: Declarative Middleware for Multimedia IPTV Services. *IEEE Communication. Magazine*, vol. 48, no. 6 (2010),74–81.
39. Soares, L. F. G. et al. Multiple exhibition devices in DTV systems. In *Proc. of the 17th ACM international conference on Multimedia*, ACM Press (2009), 281–290.
40. WAP Forum. WAG UAProf. *Technical Report WAP-248-UAPROF*-20011020-a. 2001.
41. W3C. Ora Lassila and Ralph R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation, 22 February 1999. Available at http://www.w3.org/TR/REC-rdf-syntax/

ANNEX I - Overview of NCL Elements

NCL is an XML application that follows the modularization approach. The modularization approach has been used in several W3C language recommendations. A *module* is a collection of semantically-related XML elements, attributes, and attribute's values that represents a unit of functionality. Modules are defined in coherent sets. A *language profile* is a combination of modules. Several NCL profiles have been defined. Of special interest are the profiles defined for Digital TV, the EDTVProfile (*Enhanced Digital TV Profile*) and the BDTVProfile (*Basic Digital TV Profile*). This section briefly describes the elements that compose these profiles. The complete definition of the NCL 3.0 modules for these profiles, using XML Schemas, is presented in [36]. Any ambiguity found in this text can be clarified by consulting the XML Schemas.

The basic NCL structure module defines the root element, called <ncl>, and its children elements, the <head> element and the <body> element, following the terminology adopted by other W3C standards.

The <head> element may have <importedDocumentBase>, <ruleBase>, <transitionBase> <regionBase>, <descriptorBase>, <connectorBase>, <meta>, and <metadata> elements as its children.

The <body> element may have <port>, <property>, <media>, <context>, <switch>, and <link> elements as its children. The <body> element is treated as an NCM context node. In NCM [35], the conceptual data model of NCL, a node may be a context, a switch or a media object. Context nodes may contain other NCM nodes and links. Switch nodes contain other NCM nodes. NCM nodes are represented by corresponding NCL elements.

The <media> element defines a media object specifying its type and its content location. NCL only defines how media objects are structured and related, in time and space. As a glue language, it does not restrict or prescribe the media-object content types. However, some types are defined by the language. For example: the "application/x-ncl-settings" type, specifying an object whose properties are global variables either defined by the document author or pre-defined environment variables; the "application/x-ncl-NCL" type, whose content is an embeddable NCL application; the "application/x-ncl-NCLua" type, whose content is an embeddable NCLua script; and the "application/x-ncl-time" type, specifying a special <media> element whose content is the Universal Time Coordinated (UTC).

The <context> element is responsible for the definition of context nodes. An NCM context node is a particular type of NCM composite node and is defined as containing a set of nodes and a set of links [35]. Like the <body> element, a <context> element may have <port>, <property>, <media>, <context>, <switch>, and <link> elements as its children.

The <switch> element allows for the definition of alternative document nodes (represented by <media>, <context>, and <switch> elements) to be chosen during presentation time. Test rules used in choosing the switch component to be presented are defined by <rule> or <compositeRule> elements that are grouped by the <ruleBase> element, defined as a child element of the <head> element.

The NCL Interfaces functionality allows for the definition of node interfaces that are used in relationships with other node interfaces. The <area> element allows for the definition of content anchors representing spatial portions, temporal portions, or temporal and spatial portions of a media object (<media> element) content. The <port> element specifies a composite node (<context>, <body> or <switch> element) port with its respective mapping

to an interface of one of its child components. The <property> element is used for defining a node property or a group of node properties as one of the node's interfaces. The <switchPort> element allows for the creation of <switch> element interfaces that are mapped to a set of alternative interfaces of the switch's internal nodes.

The <descriptor> element specifies temporal and spatial information needed to present each document component. The element may refer a <region> element to define the initial position of the <media> element (that is associated with the <descriptor> element) presentation in some output device. The definition of <descriptor> elements shall be included in the document head, inside the <descriptorBase> element, which specifies the set of descriptors of a document. Also inside the document <head> element, the <regionBase> element defines a set of <region> elements in a class of exhibition devices, each of which may contain another set of nested <region> elements, and so on, recursively; regions define device areas (e.g. screen windows) and are referred by <descriptor> elements, as previously mentioned.

A <causalConnector> element represents a relation that may be used for creating <link> elements in documents. In a causal relation, a condition shall be satisfied in order to trigger an action. A <link> element binds (through its <bind> elements) a node interface with connector roles, defining a spatio-temporal relationship among objects (represented by <media>, <context>, <body> or <switch> elements).

The <descriptorSwitch> element contains a set of alternative descriptors to be associated with an object. Analogous to the <switch> element, the <descriptorSwitch> choice is done during the document presentation, using test rules defined by <rule> or <compositeRule> elements.

In order to allow an entity base to incorporate another already-defined base, the <importBase> element may be used. Additionally, an NCL document may be imported through the <importNCL> element. The <importedDocumentBase> element specifies a set of imported NCL documents, and shall also be defined as a child element of the <head> element.

Some important NCL element's attributes are defined in other NCL modules. The EntityReuse module allows an NCL element to be reused. This module defines the *refer* attribute, which refers to an element URI that will be reused. Only <media>, <context>, <body> and <switch> may be reused. The KeyNavigation module provides the extensions necessary to describe focus movement operations using a control device like a remote control. Basically, the module defines attributes that may be incorporated by <descriptor> elements. The Animation module provides the extensions necessary to describe what happens when a property value is changed. The change may be instantaneous, but it may also be carried out during an explicitly declared duration, either linearly or step by step. Basically, the Animation module defines attributes that may be incorporated by actions, defined as child elements of <causalConnector> elements.

Some SMIL functionalities are also incorporated by NCL. The <transition> element and some transition attributes have the same semantics of homonym element and attributes defined in the SMIL BasicTransitions module and the SMIL TransitionModifiers module. The NCL <transitionBase> element specifies a set of transition effects, defined by <transition> elements, and shall be defined as a child element of the <head> element.

Finally, the MetaInformation module contains two elements that allow for describing NCL documents. The <meta> element specifies a single property/value pair. The <metadata>

element contains information that is also related to meta-information of the document. It acts as the root element of an RDF tree: **RDF** element and its sub-elements. (for more details, refer to W3C metadata recommendations [41]). . Meta-information does not contain content information that is used or display during a presentation. Instead, it contains information about content that is used or displayed.

ANNEX II – Data types used in the Input ControlAPI

```
<!--
XML Schema for the Input Control API data types

This is NCL
Copyright: 005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.1/ancillary/inputControlAPI.xsd
Author: TeleMidia Laboratory
Revision: 30/06/2013

Schema for the NCL media API data types.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:inputControlAPI="http://www.ncl.org.br/NCL3.1/InputControlAPI"
  targetNamespace="http://www.ncl.org.br/NCL3.1/InputControlAPI"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


   <!-- define the values for deviceTye-->
  <simpleType name="someDevicePrototype">
    <restriction base="string">
      <enumeration value="remoteControl" />
      <enumeration value="keyboard" />
      <enumeration value="motionSensor" />
      <enumeration value="positionSensor" />
    </restriction>
  </simpleType>

  <simpleType name="deviceType">
    <union memberTypes="string inputControlAPI:someDevicePrototype"/>
  </simpleType>

  <!-- declare global elements in this module -->
  <element name="device" type="inputControlAPI:deviceType"/>

  <!-- define the value for the set of keys-->
  <simpleType name="keyListType">
```

```
    <list itemType="string"/>
  </simpleType>

  <!-- declare global elements in this module -->
  <element name="keyList" type="inputControlAPI:keyListType"/>

  <complexType name="sensorType">
    <attribute name="coords" type="string" use="optional"/>
    <attribute name="key" type="string" use="optional" />
  </complexType>

  <!-- declare global elements in this module -->
  <element name="sensor" type="inputControlAPI:sensorType"/>

</schema>
```