Generic Frameworks for a Matrix of RFID Readers Based Interactions

Nicolas Géraud
Dasein Interactions,
4 pl Jean Achard, 38000 GRENOBLE
Email: nicolas.geraud@dasein-interactions.fr
Maxime Louvel
Univ. Grenoble Alpes, F-38000 Grenoble, France
CEA, LETI, MINATEC Campus, 17 rue des Martyrs, 38000 Grenoble, France
Email: maxime.louvel@cea.fr
François Pacull
Univ. Grenoble Alpes, F-38000 Grenoble, France
CEA, LETI, MINATEC Campus, 17 rue des Martyrs, 38000 Grenoble, France
Email: francois.pacull@cea.fr

***Abstract*—The paper presents first a framework to develop applications on a very innovative hardware associating hundreds of RFID readers and a high resolution display within a table. The framework is built on top of a rule-based coordination middleware that provides mechanisms to handle combinations of events, generated by the RFID readers. This framework offers the basic blocks to fully support the hardware. The paper demonstrates the interest and the possibilities of the framework through simple examples. In a second part, the flexibility of the approach is illustrated by combining this *"interface"* framework with *"rendering"* frameworks built on top of the same coordination middleware. As a result, we exemplify the re-usability approach through two scenarios (case-studies) belonging to very different application domains: help to decision making and urban mediation.**

***Keywords*-Coordination Middleware; RFID; Data aggregation.**

I. Introduction

Sensor networks are continuously growing and bringing new designs and usages. The increasing number of devices implied at the same time and the increasingly complex interactions required by the usages do not ease the task of the application programmers. There is a need for a middleware layer, offering as basic blocks high level mechanisms, in order to move most of the complexity from the application to the middleware. This paper illustrates this with an innovative smart table hosting a high resolution display and a matrix of several hundreds of RFID readers. The usage of this table is multiple when it is question of interaction, mediation and collaboration between several users. A first experience has been described in [1]. This paper goes further and shows the re-usability and extensibility of software components built with the proposed middleware. A completely different application domain is considered as a second case study.

The paper is organised as follows. Section II describes the hardware embedded in the table. The table allows to detect the identity and the position of RFID tagged objects put on the table, and to display arbitrary pictures on the HD screen. Section III presents the rule based middleware and the frameworks built on top, which offers to the application designer the basic interactions involving objects equipped with RFID tags and graphical engines managing 2D and 3D graphical objects displayed as feedback to the users. Then, Section IV puts these frameworks in context to show how interactions can be build. Section V then offers a discussion on the proposed software environment and puts it in perspective of related works. Section VI illustrates two complex applications to help decision making and urban mediation. Finally, Section VII concludes the paper.

II. Hardware

To illustrate the capability of our middleware to manage complex events detection, this paper describes our experiment with an original hardware. This hardware combines within a table, a RFID based location system and a HD screen that is used as a dynamic tablecloth.

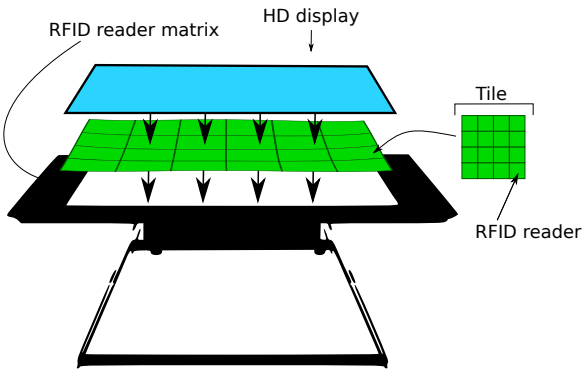

Fig. 1. Description of the table

Fig. 1 describes the table, composed of two layers. The first layer is a 42" screen able to display with a HD resolution of 1080p. This screen is seen as a classical LCD display and can thus be connected to a computer or smaller footprint board (e.g., a raspberry pi board). Under this display layer, there is a set of RFID readers organised as a matrix of 6 x 4 tiles, with each tile containing itself a matrix of 4 x 4 RFID readers. As

a result there are 24 x 16 (384) RFID readers distributed in the table. The size of a RFID reader antenna is 3.3 x 3.3 cm.

This table works with classical RFID tags that can be attached to any physical object. The raw information received for each RFID reader is the set of detected tags along with the corresponding signal strengths. This information is collected via Ethernet. Each tile has its own IP address and gives information for the 16 RFID readers constituting the tile.

There are two interesting functioning modes of this table. In the first mode (push), the tiles are autonomous and send automatically information each time a RFID is seen. In the second mode (pull), each tile can be interrogated in order to have the information corresponding to the RFID readers it contains.

With this hardware, the applicative fields are quite infinite provided that the middleware offers the required abstraction layer and a powerful mechanism to define the coordination schemes we want to put in place.

III. SOFTWARE

The presented hardware allows a lot of interactions through tangible objects. It needs a high level middleware able to quickly react to the context defined by the set of objects present on the table at the same time. Applications for this hardware typically combine RFID tag location, co-location (several tags), proximity, distance, sequence of tags put on the table. Moreover it is possible to use other interfaces connected to the system (e.g., 3d mouse, cameras). This section firstly introduces the middleware we use. For a more detailed description of this middleware, the reader may refer to [2] or [3], where it has been used in the building automation context. Then, the section presents the frameworks we developed on top to ease the creation of applications involving the table.

A. Coordination Middleware

This middleware, called LINC, is an evolution of earlier middlewares [4], [5]. LINC has been specifically re-designed to tackle lightweight embedded systems. It provides a uniform abstraction layer that eases the integration and coordination of the different components (software and hardware). It relies on the *Associative memory* paradigm implemented in our case as a distributed set of bags containing resources (tuples). Following Linda [6] approach the bags are accessed through three operations:

- `rd()` that takes as parameter a partially instantiated tuple and returns from the bag a fully instantiated tuple whose fields match the input pattern;
- `put()` that takes as parameter a fully instantiated tuple and inserts it in the bag;
- `get()` that takes as parameter a fully instantiated tuple, verifies its presence in the bag and consumes it in an atomic way.

For a matrix of RFID readers like the one in the table, bags `RawInformation` and `Position` may contain raw data such as `(tagid, tileid, readerid)` or more refined data as `(tagid, posX, posY)`. Depending on the usage (calibration or real application) they both have an interest. Once the location is computed according to the raw data, metadata may be considered from the association between `(physicalTagId, tagId)` or `(tagId, objectId)`.

The `put()` operation can insert tuples into bags configuring the readers operating mode or other configuration parameters. Finally, some bags may be used to control the videos that are displayed on the table screen.

In addition, bags can be grouped inside objects for identification purpose. For instance, the object modelling the RFID readers will contain all the bags allowing its management.

The operations `rd()`, `get()` and `put()` are used in production rules [7] to express the way these resources are used in the classical *pre-condition* and *performance* phases.

Precondition phase: It relies on a sequence of rd() operations to find and detect the presence of resources in several bags. This can be sensed values, result of service calls or states stored in tuplespaces or databases.

The particularity of the precondition phase is that:

- the result of a `rd()` operation can be used to define some fields of the subsequent `rd()` operation;
- a `rd()` is blocked until a resource corresponding to the pattern is available.

Performance phase: It combines the operations `rd()`, `get()` and `put()` to respectively verify that some resources found in the precondition phase are still present, consume some resources and insert new resources. In this phase, the operations are embedded in *distributed transactions* [8]. This ensures several properties that go beyond traditional production rules. In particular, it ensures that:

- the conditions responsible of firing the rule (precondition) are still valid in the performance phase;
- the different involved bags are effectively all accessible.

These properties are very important since they allow to verify that a set of objects are actually present "at the same time" on the table.

B. Frameworks

In LINC the approach is to define frameworks dedicated to a specific aspect of the underlying hardware or the legacy software component that is encapsulated. The frameworks provide basic blocks (LINC objects) that are then used to define the applications. The more application neutral the objects are, the more reusable they are.

Here, we consider three frameworks. The first framework is responsible for the interaction through the RFID readers embedded in the table. It has been written from scratch since it is related to a very specific hardware. It consists mainly of one LINC object called *RFID*.

The second framework is responsible for managing what is displayed on the screen. This is a standard LINC framework already used in other applications. It is composed of two objects: *`Display`* and *`2D_Engine`*. It is responsible for rendering more or less complex 2D information on a screen such as: videos, static or animated drawing, text, etc.

The third framework encapsulates off-the-self 3D engines able to render scenes containing moving 3D objects and managing textures, points of view, lights, etc. It contains two

objects *3D_OSG* and *3D_Ogre* encapsulating respectively *OpenSceneGraph* [9], [10] and *Ogre* [11], two well known 3D engines built on top of OpenGL [12]. *3D_Ogre* has been developed first, then we did the same for *3D_OSG*, keeping the same bags in order to make them interchangeable.

C. First framework: RFID Table

Object RFID: This object models the RFID readers matrix. It contains the following bags:

- `Position (tagId, posX, posY)`: contains the position of the tag (0,0 defines the top left position);
- `LogicalTag (physicalTagId, tagId)`: stores the association of a physical tagId with a more meaningful logical id, e.g., ("030209348393", "video1");
- `TagStatus(tagId, status)`: contains the status of a tag: `"in"` if detected by a RFID reader or `"out"` if not seen for a given time;
- `Mapping (tagId, objectId)`: keeps the association physical object and RFID tag that is attached to it (e.g., an hourglass used to symbolise a timer);
- `Type (tagId, type)`: maintains association of a tagId with a type of tagged object (e.g., physical object, video, action card, badge);
- `Area (areaId, areaDefinition)`: contains areas on the table defined as a set of points forming a polygon;
- `PositionArea (tagId, areaId)`: contains the tagId contained in a given area.

The detection of the tags placed on the table is done by a driver that handles the events sent by the different RFID readers (used in push mode). This information is decoded and the different bags are filled with the corresponding resources. When a tag is detected, the driver computes its position on the table (X,Y) and adds the resource `(tagId, posX, posY)` in the bag `Position`. A tag is detected by one or several RFID readers of the table. To improve the precision of the detected location we can use the signal strength provided by the readers. Thus, we have higher precision than the size of an RFID reader antenna. Practically, we can consider a step equal to the third of the antenna size (around 1.1 cm).

As a RFID reader continuously sends the tag information and as the information slightly vary, a filtering is applied to avoid inserting new resources when it is not necessary. Hence, a resource is inserted only when a significant change in the location is effective. In addition, the `status` of the tag, `"in"` if the tag is still on the table or `"out"` if it has been removed (i.e., not be seen for a given time), is inserted as a resource `(tagId, status)` in the bag `TagStatus` each time the `status` changes.

The bags `Type`, `Area` or `LogicalTag` are configuration bags and their usage is described here after.

Introduction to rules: The described middleware allows to express with its rule based language actions to be performed (performance phase) when some conditions (precondition phase) are verified. The actions performed are embedded in transactions enclosed in {}. As `rd()` actions may be included in these transactions, it is possible to ensure that resources found in the precondition are still valid in the performance.

```
1 ["RFID","TagStatus"].rd(tagId, "in") &
2 # other preconditions
3 ::
4 {
5  ["RFID","TagStatus"].rd(tagId, "in");
6    # other actions
7 }.
```

Listing 1. Ensure tag is still there at performance phase

Listing 1 presents an example of rule, where the precondition and performance part are respectively before and after the `"::"`. To simplify the example, we only show a single operation in the precondition and performance phase but both may contain several additional tokens.

The first token (line 1) reads in the bag `TagStatus` of the object `RFID` all the tags with status "in". This allows to detect new tags placed on the table and then to manage the corresponding scenario. Line 5 guaranties that the `tagId` is still on the table when the performance phase is executed. Since actions in the performance are embedded in transactions the other actions can only be done if the tag is still there. Note that this approach simplifies a lot the management of events:

- events are detected in preconditions;
- when performances are executed, guarantying that the condition related to the event is still valid only requires to add a `rd()` in the performance part.

Initialisation rules: Listing 2 presents an initialisation rule. No precondition is defined, this rule is always executed and only once at the application launch time.

```
1 ::
2 {
3  ["RFID","LogicalTag"].put("9e7f9cce9","tag_video_table");
4  ["RFID","Area"].put("zoneA","0,0;0,54,12,54;12,66;66,0");
5  ["RFID","Type"].put("t_video_presentation_table","video");
6 }.
```

Listing 2. Initialisation rule

Here we initialise the bags `LogicalTag`, `Area` and `Type`.

In the first bag, we associate the physical tagid `"94e7f89cce9"` to the more user friendly logical tag `"tag_video_table"`. This allows to manipulate in the rules an id that is human readable. In addition, several physical tags can be associated to the same logical tag for backup reason or to offer to several people the possibility to trigger the same action with different objects or cards.

In the second bag, we define a `"zoneA"` as a list of points defining a polygon. This is taken into account by the driver to populate the `PositionArea` bag.

In the third bag, we associate a `type` to a tag. The `type` allows to define a specific context around this tag to verify that it is correctly used. For instance, a tag associated to a voting card cannot be placed everywhere on the table but in a given area. Another usage is to give information to the driver about the sampling frequency for a given tag or if the change in the location is large enough to be reported or not.

Defining action area: To better organise the table, area (i.e., zone of the table) can be used. An area is defined by adding a resource `(areaId, areaDefinition)` in the bag `Area`. The `areaDefinition` is a set of points defining a polygon. When the RFID driver detects a new position for a tag, at the same time it inserts the corresponding resource

in the bag `Position`, it scans all the defined areas and adds the resources `(tagId, areaId)` in the bag `Area`. In the same manner, when the driver inserts a resource `(tagId, "out")` in the bag `Status` it removes all the resources corresponding to the tag in the bag `Area`. This simplifies the application designer's task since she can directly write a rule that starts with a token reading in the bag `Area`.

D. Second framework: 2D Rendering Engine

This framework is a generic 2D rendering engine. Its role is to manage what is displayed on a screen. It includes an object more oriented to video rendering and another that display arbitrary 2D fixed or animated graphical objects. The target, can be, as in our case, the screen included in the table, but also a smart TV, a regular computer screen, a tablet or a smartphone.

Object `Display`: The first object of the framework manages the displays on the screen. It contains the following bags (non exhaustive list):

- `videoPlayer (playerId, videoname, posX, posY, width, height, orientation, soundTrack)`: this bag accepts only the `put()` operations and launches a video player displaying the video corresponding to the filename with the given geometry, with or without sound track;
- `video (videoname, status)`: maintains the status of the video among `started`, `finished`, `paused`;
- `videoPlayerCommand(videoname, command)`: this bag accepts only the `put()` operations and the following commands: `"stop"`, `"pause"`, `"resume"`, `"fs_on"`, `"fs_off"` (fs is for full screen).

A simple usage of this object is described in Listing 3. This initialisation rule starts the video called *video_table* presenting the table on the top left corner of the screen. When the performance is executing, a video player is started and configured to display the video with the resolution (640x480) at position (0,0). The status of the video is set to `"started"`.

```
::
{
 ["Display","video"].put("video_table","started");
 ["Display","videoPlayer"].put("vlc","video_table",
      "0","0","640","480", "True");
}.
```

Listing 3. Start presentation video of the table

To easily support any kind of video player, the framework uses an external Linux process. The role of this process is to display a video according to a media definition file containing the basic information needed to define the layout, the position, the fact that the sound track is on or off. The display driver saves the PID of the process started in order to interact with it independently of the video player used.

Listing 4 shows how to stop a video. The precondition waits that the stop card is placed on the table. It then reads the `videoId` of the started video. The performance actually stops the video just by sending the signal `SIGKILL` to the PID playing the video.

```
["RFID","TagStatus"].rd("tag_stop_video", "in")&
["Display","video"].rd(videoId,"started")
::
{
 ["Display","videoPlayerCommand"].put(videoId, "stop");
}.
```

Listing 4. Stopping a video with a control card

Object `2D engine`: The second object of the framework is a 2D engine. It is in charge of displaying the background of the table. It is also in charge of the displayed animations. The current version relies on a Scalable Vector Graphics [13] (SVG) engine to define 2D animations that will be displayed in a simple web browser that is opened in full screen on the table display.

The 2D engine object contains the following bags (non exhaustive list):

- `Background (imagefile)`: When a resource (i.e., an image file) is inserted it replaces the current background of the table;
- `Media (tagId, filename)`: associates a `tagId` to a `filename`;
- `Sprites (spriteId,x,y, svgfile)`: Allows to display a sprite (SVG image) at the position `x,y` on the table screen;
- `MoveSpriteGrid (spriteid,x,y,duration, nbsteps,renderlist)`: Allows to define an animation for the sprite defined by `spriteid`. The `duration` of the animation using; `nbsteps` steps and using successively the SVG patterns defined in `renderlist`;
- `Visibility (spriteId,percent)`: defines the opacity and the visibility of the sprite.

This object actually contains more bags that allow not only to define a background but also sprites that can be animated on top of this background. All the SVG attributes may be dynamically modified.

The animations are done at the level of the object that returns an html file when invoked through URL. This HTML file is built from static information (HTML [14] and SVG [13] templates) present in the file system and contextual information present in the bags.

In addition, SVG templates are filled by javascripts [15] to bring the dynamic aspects through animated SVG entities. Finally, through SVG and javascripts it is possible to attach URL based interactions to classical events *mouseclick*, *mouseover*, etc. These URL calls either insert or read resources in bags dedicated to user interactions. Resources are put in bags to capture the inputs from the users. Furthermore, resources are regularly read in bags to obtain updated information. The resources are returned to the web browser as json structure [16] easily understandable by javascripts. Thus, with devices allowing user interactions (e.g., tablet and phone) we can go very far in term of user interface.

The main advantage of this approach is that we use the full power of nowadays web technologies without paying an heavy cost at the rendering level. As most of the current equipments are surprisingly able to manage quite complex web pages, this framework can deal with almost all the user life

objects owning a screen. For instance, we have built a user interface including SVG drawing for home automation with a simple kindle paperwhite e-reader. Thus, you can have a tablet-based interface always available in your living room with an autonomy of a month.

E. Third Framework: 3D Rendering

In the same way, we have developed a generic 3D rendering engine that allows to display arbitrary 3D scenes.

Object `3D engine`: This object encapsulates the 3D engine Open Scene Graph [10]. It runs on a multicore laptop and the output is displayed either on an external large screen or a video projector. The role of this object is to display complex interactive 3D scenes. To deal with performance, scalability and high quality user experience the LINC object has been decomposed in two parts. First, we have a set of bags that are used to store the basic information about the manipulated entities.

- `Entity (entity, file)`: contains 3D models of buildings that are the same as the ones used for printing the buildings on the 3D printer. Thus, no extra effort is required for the 3D virtual scene.
- `Light (aspect, r, g, b)`: this bag allows to insert different types of light that are used for rendering the scene.
- `Mode (key, mode)`: this bag allows to define the visualisation modes. Currently, we consider objective or subjective view.
- `Location (entity, x, y, z)`: this bag keeps tracks of the location of the different entities displayed in the 3D scene.

In addition, we have a bag `Command (command, entity, p1, p2, p3)` that is associated to the 3D engine launched as an independent process. This bag is regularly interrogated by the 3D engine and the commands are collected one by one and executed in the context of the 3D scene. The reason of such architecture is first to dedicate one of the CPU cores to OpenSceneGraph managing the 3D scene. Second, changes are only done when an update of the scene is required, if no command is present nothing needs to be recomputed and changed. Third, this allows the 3D engine to pace the rhythm of command executions. If the rendering is very complex then the frequency of update will slow down accordingly, the bag working as a buffer. Thus, the user experience does not suffer of the possible saturation of the 3D engine.

To deal with priorities, we do not consider a single `Command` bag but two. The second bag is associated to high priority commands. Thus, for instance, we can manage in priority the commands linked to the displacement of the user in a subjective view while adding new 3D objects in the background scene is managed when possible.

F. Other available frameworks

Other frameworks, not used in this paper use cases, have been built on top of the LINC middleware. Following the same patterns, objects are linked to a dedicated context. They define specific frameworks ready to be used to design new applications.

For instance, to integrate sensors and actuators we have built a framework that considers the main standards for wired and wireless technologies. Each technology is managed by a dedicated object acting as a gateway. All the objects share the same set of bags to hide the heterogeneity. In addition, we have developed another framework to integrate camera (movement detection, face detection), light systems or mobile robots. We also developed a framework for managing the dynamic creation of scenarios and their management in term of context and priorities. The three of them have been used in the context of building automation [3], [17]. Finally, we have also recently been working on a voice framework to integrate voice recognition and text to speech engine.

IV. Frameworks in context

After presenting the global architecture, this section shows how interactions involving objects from different frameworks can be easily encoded with rules, through several examples.

A. Architecture

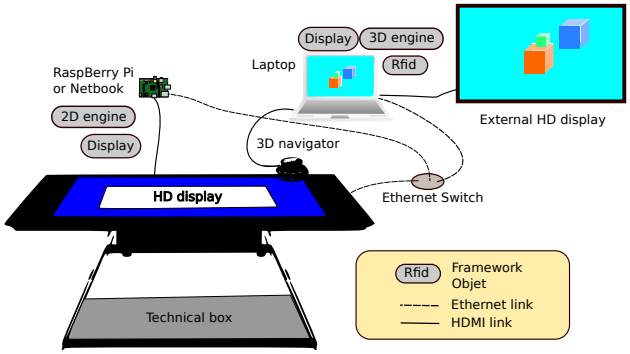

Fig. 2. Global picture.

Fig. 2 presents the current hardware and software setting. It contains the table described in Section II. In addition, there are two computing resources: a laptop and a raspberry pi or a netbook depending on the complexity we want to manage on the screen table. In real situation, they are embedded inside the technical box at the base of the table, hidden from the users. The table's screen is connected to the raspberry pi with a HDMI cable, offering a 1080p HD resolution. The Ethernet switch defines a local area network connecting the matrix of RFID readers, the raspberry pi/netbook and the laptop. From the software point of view, the objects of the different frameworks are distributed among the two computing resources.

The software configuration can be changed according to the application. For instance, this paper details two applications (in Section VI).

For the first application, the `RFID` and `Display` objects run on the laptop, the `2D engine` and another `Display` objects run on the Raspberry pi. For the second application, in addition a `3D engine` object runs on the laptop. The

raspberry pi is replaced by a netbook to use more intensively the `2D engine` and the `Display` objects.

Some LINC objects launch background processes that interact with them. For instance, the `2D engine` object launches a web browser, displayed in full screen, responsible for displaying the generated HTML + SVG files. The object `3D engine` launches OpenSceneGraph engine also in fullscreen. Finally, the object `Display` launches on demand one or more instances of VLC multimedia viewer.

B. Examples of simple interactions through rules

To change the background with a card: In this example (Listing 5), the background displayed on the screen is changed when a card of type `background` is put anywhere on the table. The first line of the precondition makes the rule fire only for tags that are known to define a background. Then (line 2), whenever a card of this type is on the table, line 3 finds the `filename` image corresponding to the `tagid`. Then, in the the performance phase the action (line 6) changes the background of the table with the content of filename. After the change, it is not necessary to let the card on the table.

```
1 ["RFID","Type"].rd(tagId, "background")&
2 ["RFID","TagStatus"].rd(tagId, "in")&
3 ["2D_Engine","Media"].rd(tagId, filename)
4 ::
5 {
6  ["2D_Engine","background"].put(filename);
7 }.
```

Listing 5. Rule to change the background when a card is put on the table

You can define as many backgrounds as you want, you just need to insert a resource defining the type of your RFID tag as a `"background"` in the bag `"Type"` and a resource to associate the tag id to an image filename in the bag `"Media"`.

To display a video at the location defined by the card: This example aims at starting a video when a card of type `video` is put. The card's position defines the top-left corner of the video. Listing 6 gives the rule implementing this scenario. As previously, lines 1 and 2 make the rules fires when a video card is put on the table. Line 3 gives the card's position. Finally, line 4 finds the video to be displayed from the tag id. The performance then embeds in one transaction:

- ensuring that the card is still there (line 7);
- starting of the video player with (posX,posY) (line 8);
- saving state "started" for the video (line 9).

```
1  ["RFID","Type"].rd(tagId, "video")&
2  ["RFID","TagStatus"].rd(tagId,"in")&
3  ["RFID","Position"].rd(tagId, posX, posY) &
4  ["RFID","Mapping"].rd(tagId, videoId) &
5  ::
6  {
7   ["RFID","TagStatus"].rd(tagId, "in");
8   ["Display","videoPlayer"].put("vlc", videoId, posX,
        posY, "640", "480", "True");
9   ["Display","videoPlayer"].put(videoId, "started");
10 }.
```

Listing 6. Display video at card's position

To switch a video currently played as a full screen video: This scenario uses in addition to the card that started a video as in previous scenario, a card that defines the modality `full-screen`.

Listing 7 implements such scenario. Lines 1-3 check that both cards are on the table. Line 4 fires when the video has been started (by rule in Listing 6). The performance phase checks that both cards are still on the table and the video is still in started state. In these conditions, we switch the video player to full screen by adding a resource in the bag `videoPlayerCommand` (line 10).

```
1  ["RFID","TagStatus"].rd("tag_fullScreen", "in")&
2  ["RFID","Type"].rd(videoId, "video") &
3  ["RFID","TagStatus"].rd(videoId, "in")&
4  ["Display","videoPlayer"].rd(videoId, "started")&
5  ::
6  {
7   ["RFID", "TagStatus".rd(videoId, "in") ;
8   ["RFID","TagStatus"].rd("tag_fullScreen", "in");
9   ["Display","videoPlayer"].rd(videoId, "started");
10  ["Display","videoPlayerCommand"].put(player, "fs_on");
11 }.
```

Listing 7. Put video in full-screen

It is then necessary to write a rule (Listing 8) to quit the full screen mode if the full screen card is removed from the table. This rule is triggered when the full screen control card is `"out"` (line 1). As previously, the performance checks the player and the cards' status and adds a resource in the bag `videoPlayerCommand` (line 10).

```
1  ["RFID","TagStatus"].rd("tag_fullScreen", "out")&
2  ["RFID","Type"].rd(videoId, "video") &
3  ["RFID","TagStatus"].rd(videoId, "in")&
4  ["Display","VideoPlayer"].rd(videoId, "started")&
5  ::
6  {
7   ["RFID", "TagStatus".rd(videoId, "in") ;
8   ["RFID","TagStatus"].rd("tag_fullScreen", "out");
9   ["Display","VideoPlayer"].rd(videoId, "started");
10  ["Display","VideoPlayerCommand"].put(player,"fs_off");
11 }.
```

Listing 8. Quit full-screen

V. Discussion and related works

So far we have illustrated the simplicity of writing interactions with the proposed frameworks. The detailed examples showed how information coming from distributed sources may be aggregated as a complex distributed event.

In the literature such task is usually implemented with a publish-subscribe approach [18], where subscribers register to specific events generated by publishers (RFID readers in this paper). This has been applied for instance in the context of sensor networks [19]. With such a system it is possible to write code that would be similar to the precondition part of the rules presented in this section. For instance, to react to a tag detected in a specific area or to an external event. However, in a publish-subscribe approach when the system has to react upon a set of events or to be sure that the events are still valid when the actions have to be executed, the amount of additional code is not negligible.

With the framework developed on top of our middleware expressing an event as "one card is put in a specific area" and "another card of a specific type is put at the same time anywhere else" is simply a sequence of `rd()` tokens. In addition, defining what to do if a card is put on the table and immediately removed is possible thanks to the distributed transaction offered in the performance phase that

aborts a transaction if the conditions are not currently true. In a publish/subscribe approach it would correspond to remove events that are no longer true but are still present in the system. Dealing with that is not impossible, but it is for sure not an easy task.

Other works have used as us resources and tuple spaces to facilitate the coordination in a distributed system such as [20] or [21]. These work focus on providing context awareness to mobile applications. However, we believe that these approaches still rely too much on traditional object oriented paradigms to be flexible enough to address the hardware used in this paper. Other promising approaches for coordination of complex systems have been proposed in [22] and [23]. For instance, in [22], the authors propose to use the chemical reaction paradigm to model the tuples evolution. The basic idea is to let the tuples evolve and auto-regulate as in chemical reaction. Even though these approaches could help coping with complexity of interactions brought with the table, we see two limitations. Firstly, self evolving coordination may be too complex to build specific interactions such as the ones described in this paper. Secondly, this work seems to be only at the theoretical level since no implementation has been done.

VI. Applications

This section presents two applications using the table: the first one helps decision making, the second deals with urban mediation. In both cases, we show how the frameworks detailed previously have been used to build the applications. The main advantage of our approach is that the objects of the frameworks are very generic and thus highly reusable. The specificities of the applications are in the coordination rules.

A. Help for decision making

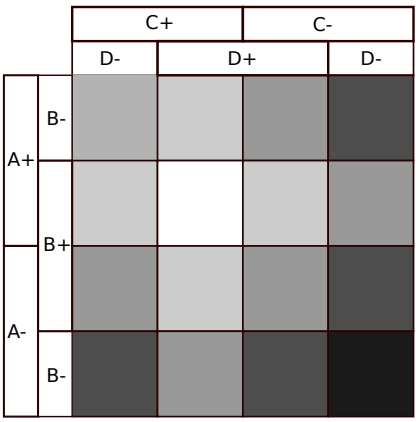

Fig. 3. Veitch Diagram

The first application uses quite complex interactions between several users around this table. The application allows to collect the opinion of a panel of people to elect the best equipment, concept or decision according to a set of criteria. In the present example, the panelists are asked to give their opinions about a set of smartphones according to the following criteria: aesthetic, user interface, size and autonomy. These criteria are denoted A, B, C and D, respectively, and can take the value positive or negative depending on the majority of votes from the panelists. The resulting information is displayed as a Veitch diagram as shown in Fig. 3.

The white cell contains the best choices that received 4 positive opinions. The adjacent cell contain choices that received 3 positive opinions. The darker a cell is, the more negative it is. The black bottom right cell contains the worst choice with 4 negative opinions. This section now details what is an interaction session and gives a few hints on the implementation.

1) Interactions: At the beginning, each panellist has a badge representing his/her identity. The master of session presents a smartphone and may display a video on the table by putting the corresponding card on it. Some modifier cards added to the table may modify the display either by switching to fullscreen or by launching a second video player with a 180 rotation to adapt to situation where people are all around the table. In this case, the second video uses the same video flow (without sound track) and is synchronised with the first one. Once the presentation is done, the vote may start. The master of session places the card corresponding to the criterion (e.g., aesthetic) and then the table display shows two areas, one green to collect the badges of panelists liking the smartphone design and one red for those that are not enthusiastic. An additional video or photo specific to this criterion may also be displayed. Then, the master of session triggers the vote by placing an hourglass (tagged with an RFID) on the table. A timer indicates at each corner of the table the remaining time for the vote. Each panellist put her badge on the table according to her opinion. A circle is displayed around the badge to return a feedback to the user. Different modalities may be configured at the beginning of the session to control the vote:

- the duration of a vote phase;
- missing vote is considered as negative or positive;
- a vote is definitive or not.

Once the timer reaches zero the votes are stored for further processing and the master of session can go to the next criterion. When all the criteria have been considered, the master of session can go to the next smartphone. At any moment, the master of session may place a card on the table to display or print current the status of the Veitch diagram.

2) Implementation: The full application described here may be implemented by using small variation of the basic interactions involving `Display`, `2D_Engine` and `Rfid` objects presented previously.

A specific `Application` object has been added. It contains bags to store panelists identities, votes and current step in the session (smartphone number and criteria number). This object is dedicated to the application and is the only one that is not re-usable. Associated with the coordination rules, this defines the logic of the applications. All the other objects are just a mean to access to the external world: table, screen, etc.

The basic settings, configuring a working session, are done through initialisation rules that define modalities such as default value of missing vote or the identity of the panelists. Note that using the proposed framework is very appropriate

since adding new features simply requires to add new rules. Existing rules can continue to work without concern. For instance, we can use an initial round getting the identities of the panelists rather than using a configuration rule. This can be done without any other impact than replacing the initialisation rule with the following rule required to obtain the identity of the panelists.

Registration: The following rule (Listing 9) starts when the *registration card* is placed on the table. It basically stores the `tagId` of all the users who placed their id badge on the table, in the bag `Users` of the object `Application`.

```
["RFID","Status"].rd("tag_registration","in") &
["RFID","Status"].rd(tagId,"in") &
["RFID","Type"].rd(tagId,"badge") &
::
{
 ["RFID","Status"].rd("tag_registration","in") &
 ["RFID","Status"].rd(tagId,"in") &
 ["Application","Users"].put(tagId)
}.
```

Listing 9. Registration

When the *registration card* is removed, the resource `("tag_registration","in")` is removed from the bag `Status` replaced by `("tag_registration","out")`. This immediately stops the effect of the registration rule: performances will fail on the first token because of the absence of the resource.

As a result, when the card is removed, all the registered users are in the bag `Users` of the object `Application`.

Vote: A session of vote would then start with the following rule:

```
["RFID","Status"].rd("tag_vote","in") &
::
{
 ["Application","Step"].put("vote_started")
 ["2D_Engine","Background"].put("vote_bg")
}.
```

Listing 10. Vote starting

It defines that we are in the step `"vote_started"` and changes the table background creating a zone for the positive votes in green and a zone for the negative votes in red.

The vote round stops either when the delay associated to the round has expired with the following rule:

```
["Application","Step"].rd("vote_started") &
SLEEP: 30
::
{
 ["Application","Step"].get("vote_started")
 ["2D_Engine","Background"].put("end_vote_bg")
}.
```

Listing 11. Vote end (time out)

or when the master decides the end of the round, for instance because everybody have voted. This is done when she removes the cards associated to the `tag_vote`.

```
["Application","Step"].rd("vote_started") &
["RFID","Status"].rd("tag_vote","out") &
::
{
 ["Application","Step"].get("vote_started")
 ["RFID","Status"].rd("tag_vote","out") &
 ["2D_Engine","Background"].put("end_vote_bg")
}.
```

Listing 12. Vote end (master decision)

At the application level it does not matter which of the 2 rules has been applied. As both of them consume the resource `vote_started` the execution of one will prevent the other to be executed.

The combination of the two rules defines that we are no longer in the step `"vote_started"` and changes the table background to indicate the end of the round.

The rule managing the vote itself is given in Listing 13.

```
["Application","Step"].rd("Vote_started") &
["RFID","Status"].rd(tag_product,"in") &
["RFID","Type"].rd(tag_product,"product") &
["RFID","Status"].rd(tag_property,"in") &
["RFID","Type"].rd(tag_property,"property") &
["Application","Users"].rd(user_tagId) &
["RFID","Status"].rd(user_tagId,"in") &
::
{
 ["Application","Step"].rd("vote_started")
 ["RFID","Status"].rd(tag_product,"in")
 ["RFID","Status"].rd(tag_property,"in")
 ["RFID","Status"].rd(user_tagId,"in")
 ["RFID","Area"].rd("positive",user_tagId)
 ["Application","Vote"].put(tag_product,
     tag_property,user_tagId, "+")
}
{
 ["Application","Step"].rd("vote_started")
 ["RFID","Status"].rd(tag_product,"in")
 ["RFID","Status"].rd(tag_property,"in")
 ["RFID","Status"].rd(user_tagId,"in")
 ["RFID","Area"].rd("negative",user_tagId)
 ["Application","Vote"].put(tag_product,
     tag_property,user_tagId, "-")
}.
```

Listing 13. vote

The first token of the precondition and of each transaction in the performance phase is a guard ensuring the rule is only active during the vote. (lines 1, 10 and 18) When the vote ends, the resource `("vote_started")` is removed from the bag `Step`. As a result, all the transactions will fail on the action `rd("vote_started")`. The `rd()` operations on `(tag_product,"in")` and `(tag_property,"in")` (lines 2 and 4 in the precondition) ensure that the vote is considered for the current property of the current product. Token in line 6 of the precondition returns all the users registered for the vote. This is a very natural manner to avoid considering votes by an unregistered person. Obviously, as this information is stored in a bag, it would be possible at any time to register or unregister a user. Finally, the last token of the precondition waits for each user tag to be put on the table. For each of these tags, a performance is triggered. The performance is composed of two very similar transactions. The first four tokens (lines 10 to 13 for the first transaction) are guards to ensure that the vote is still open, for the current product, the current property and that the user tag is still on the table. The last two tokens of the transactions define the vote and save it. A vote card can be in only one area, and the two areas cover all the table. Then, the two transactions are exclusive. The first transaction succeeds for positive votes, while the second transaction succeeds for negative votes (enforced by lines 14 and 22). Hence we ensure that one and only one transaction will succeed, counting the vote correctly.

When all the votes have been done for a criterion, the users can remove their voting card.

This rule applies for all the products and the criteria inside a product since the context is given by the corresponding cards placed on the table.

Then, the master simply removes the criterion card and replaces it by the card for the next criterion. This will trigger a new branch in the precondition, at line 4. The value of `tag_property` now contains the value of the new property. Once all the criteria have been voted for a product, the master replaces the product card by another product card and the vote may continues with a new round of properties.

The goal of this section was not to describe all the rules involved in the application but to show how problems that seems quite complex may be managed with only a few number of rules. Moreover, as the application can be decomposed in steps and each rule associated to a step is controlled by the presence of some specific resources in some bags, we can easily avoid the "unwanted" competition between rules. Thus, it is very easy to guaranty that a set of rules verified in isolation will not introduce flows in the whole set of rules defining the full application.

B. Urban Mediation

The context of this second application is the mediation in between people who design urban infrastructures, deciders of the project and inhabitants who live or will live inside or nearby the area. The table is used both as a ground for a physical model of the urban sector under consideration and as an interaction medium to augment the model with 3D virtual representation of the same urban scene as shown in Fig. 4.

Physical building models can be put on the table to figure out the area to be considered. These buildings are made with a 3D printer and they are equipped with Rfid tags to allow interaction with the table.

It is possible thanks to the framework `2D Rendering` to display various useful information on the table screen.

- Static information:
  - maps or aerial view;
  - parking lots, road, green spaces;
  - electricity, communication, water or drain networks;
  - underground transportation;
  - meta information such as reserved location;
  - specific difficulties linked to the ground nature (floodplain, rocky soil, historic relic).
- dynamic information:
  - graphs and statistical information;
  - animation to render wind direction (venturi effect), sound propagation;
  - historical traffic data;
  - videos.

An additional external screen or a projector is used to render the urban area as a virtual 3D scene. This allows the user to either have an objective view, flying over the full scene or a subjective view, moving around directly within the scene. This uses the framework `3D Rendering`. In Fig. 4, an objective view of the area is displayed.

This section now details how the user may interact with the application and then gives some hints on the implementation.

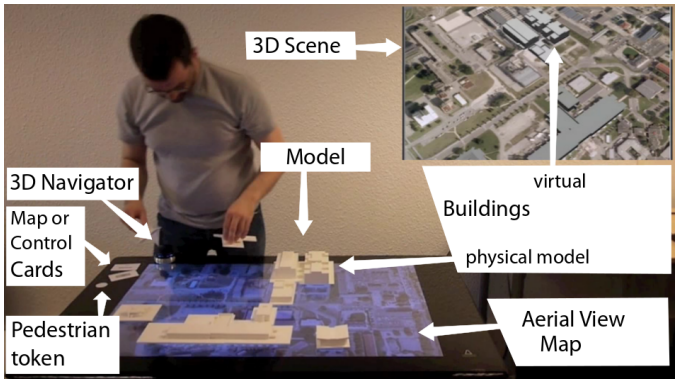

Fig. 4. Urban mediation.

1) Interaction: In this application we do not have a strict succession of steps as in the previous application. On the contrary, we just have different *"modes"* that can be freely set at any moment by the user through the use of a context card placed on the table.

For instance, a mode populates the physical model (on the table) and thus the virtual scene (on the wall) with buildings and other urban elements.

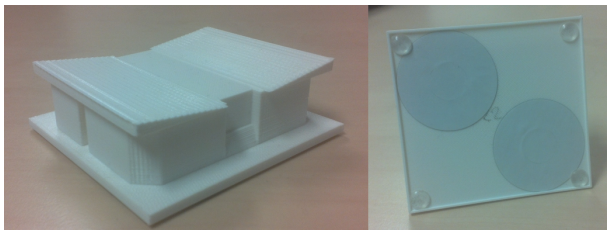

Fig. 5. Printed building and its verso equipped with RFID tags.

This is done by putting on the table 3D building models as the one shown in Fig. 5. Each building has on the verso two RFID tags placed respectively in the top left and bottom right corners. These tags are used for the identification, the location on the table and the orientation of the building.

Given the respective positions of the two tags, the barycentre corresponds to the centre of the building that is required to compute the actual position in the virtual scene. Using two tags per building model also allows to have a rough idea of the orientation of the building. Given the resolution of the table we can consider steps of 45 degrees.

We define a single logical tag (concatenation of the two physical ones) that uniquely identifies the building. Then, we associate this logical tag to the physical building model, its 3D model file, its location, its orientation and its virtual representation in the 3D scene. This bridges the physical world and the virtual one.

The 3D printed shapes represent, in the current urban scene, the real buildings and the buildings for which the plans are already known. However, sometime, for discussion in the early phases, we want to have an idea of the impact of a possible building. In this case, we use different black boxes as shown the Fig. 6 prefiguring buildings of different shapes and sizes. These blocks are also equipped with two tags.

Fig. 6. Generic shapes for prefiguring future buildings footprint.

In order to enrich the urban scene, we can also use some *"ink pads"* that once applied on the table create 2D markers on the background and elements in the 3D virtual scene. This concerns for instance trees, hedgerow and other scenery elements.

The background can also be modified by placing cards on the table. Backgrounds can be superposed in order to reveal various additional informations as described in the previous section.

It is also possible to move a token representing a pedestrian within the physical model and acting on her field of vision with the 3D Navigator. This is a device that includes the functionality of a classical joystick in X and Y combined with informations push and pull to add the Z axis. In addition, you can rotate it and you have one button on each side.

As a result, we can see on an external screen, the 3D virtual scene actually seen by the pedestrian. Moving the token allows to progress in the 3D scene, visiting the different places. Rotation of the pedestrian is done through the 3D navigator. The pedestrian can thus turn on herself getting a 360 view. A vision cone is displayed on the table to show the current pedestrian field of view. The same device is used to manage the head movement up, down, right and left. Finally, the two buttons on the right and left sides of the device allow the pedestrian to respectively go up and down. This allows to consider the view from the different floors of a building. The corresponding floor number is displayed on the table near the vision cone.

All the modifications of the physical model are immediately echoed on the 3D virtual scene and thus, the potential impact can be better evaluated.

Switching back to objective view is done via a special card placed on the table. In this mode, the 3D navigator is used to fly over the scene with the same capability of movements.

In objective mode, it is also possible to enrich the 3D scene according to lights and sun position. It is thus possible to see the impact of a building in term of shadow in the neighbouring according to the season, the hour, etc.

It is also possible to associate a texture to the black boxes to test alternative to a project and how the future building can be better integrated with the existing ones.

Finally, it is possible to launch video on the table screen or on the external one.

2) Implementation: The application uses a combination of the three frameworks described in this paper. These frameworks provides very generic and reusable objects: a large part of them is already used in the previous application. Some of the interactions and the application logic of this second application is given in the following as illustration.

The first rule manages the 3D printed buildings.

```
["RFID","TagType"].rd(id,"building") &
["RFID","TagPosition"].rd(id,x,y) &
COMPUTE: x1, y1, z1 = transform(x, y) &
["OSG","Entity"].rd(id,filename) ;
::
{
 ["OSG","Command"].put("entity",id,filename,"","") ;
 ["OSG","Command"].put("translate",id,x1,y1,z1) ;
 ["OSG","Command"].put("scale",id,"1","1","1") ;
}.
```

Listing 14. Buildings

The precondition waits for all the tags of type `building` in order to manage only tags corresponding to the 3D printed building models. It then reads the location of the model on the table. From the 2D coordinates it computes the corresponding 3D coordinates in the virtual scene through the method `transform()`. The method `transform()` is written in python [24] code and may be very simple if we just do a simple translation for `x` and `y` and set `z` to 0. It can be more complex if we refer to a model of the ground that in addition can give the `z` coordinate according to `x` and `y`. The last token allows to obtain the 3D model file associated to the building.

The performance phase then inserts in the `Command` bag the three commands required to create in 3D virtual scene with the representation of the building. The first one creates the entity according to its 3D model if it is not already done. The second places the entity at the right place. The third changes the scale of the object in the 3 dimensions. Indeed, by default, when a new graphical object is created it has an initial scale of 0 to stay hidden. Then we can make all the transformations (translate, rotate, etc.) before acting on the scale in order to make it appears at the correct place.

Moving a building will trigger another instance of this rule with a new resource for the tag position and the building will be moved accordingly in the 3D virtual scene. Removing a building is detected by the status of the tag that becomes `out` and in this case, we just set the scale of the object to 0 in order to hide the building.

The second rule manages scenery elements that can be added to the 3D virtual scene. It is almost similar to the previously presented rule concerning buildings. The main difference is that we have not one physical tagged object per virtual one. Thus, we use a tag equipped *"ink pad"* for each type of element we want to consider. In the following example, we consider a small tree. Each time this the pad is put on the table, a new instance of the small tree is created as if a physical 3D object where placed. As a result, a 3D virtual small tree appears at the corresponding location in the 3D virtual scene.

```
["RFID","TagPosition"].rd("small_tree",x,y) &
COMPUTE: x1, y1, z1 = transform(x, y) &
COMPUTE: id = "elt_"+x+"_"+y &
::
{
 ["OSG","Entity"].put(id,"arbre.osg") ;
 ["OSG","Command"].put("entity",id,"tree1.osg","","") ;
 ["OSG","Command"].put("translate",id,x1,y1,z1) ;
 ["OSG","Command"].put("scale",id,"1","1","1") ;
}.
```

Listing 15. Small tree ink pad

The precondition phase reads the 2D coordinates of the pad on the table. Each pad has an unique identifier, here it is `small_tree`. The 3D coordinates are computed as in the previous rule. The last token creates an `id` that will be associated to the virtual entity that is going to be created. This entity is in fact an instance of the type of tree associated to the pad. The id is of the form `elt_<x>_<y>` and is unique for a given location. Indeed, we consider that we can have a single scenery element at a given place.

In the performance phase, we declare a new entity referred by its `id` and associated to it the corresponding 3D model here in the format `.osg` native to OpenSceneGraph. We then insert in the `Command` bag the three same commands as the previous rule to make the trees appear in the scene.

This rule is very generic and we can manage, with small variants, pads associated to various scenery elements. For instance, we can act on the 3D model by using another model file or we can act on the size by using different scale factors. In LINC it is possible to generate dynamically new rules, thus it is very simple to add new ink pad on the fly via a simple web interface.

To remove an object created by a tag equipped ink pad, we can use a tag equipped rubber as shown in the following rule.

```
["RFID","TagPosition"].rd("rubber",x,y) &
COMPUTE: id = "elt_"+x+"_"+y &
["OSG","Entity"].rd(id,filename) &
::
{
 ["RFID","TagPosition"].get("rubber",x,y) &
 ["OSG","Entity"].get(entityname,filename) ;
 ["OSG","Command"].put("scale",entityname,"0","0","0") ;
}.
```

Listing 16. Rubber

Each time the rubber is detected at a place, we compute (line 2) the `id` of a potential element that would have been created by a pad. If the entity exists (line 3) then we can go further and remove the element. This is done in the performance part where we consume the information about the presence of the rubber that is no longer required. Keeping this information would prevent to put later another element at this place, because it would be automatically immediately removed. The entity itself is removed from the corresponding bag and finally we hide the virtual element of the 3D scene by setting a scale to 0. This creates an orphan hidden virtual element that will be garbaged if we create a new scenery entity at the same place since it will have the same id.

If the rubber is placed on an empty location, nothing is done and when it is removed from the table the resource `("rubber",<x>,<y>)` is removed from the bag `TagPosition` preventing any erase action to be done at this place.

The full application contains two tens of rules following more or less the same scheme. Inputs from the table and the 3D navigator are combined and used to act on the table screen and/or the 3D virtual scene. Here too, as the scenario depends on a specific combination of tags read by the table, we can ensure that each scenario is guaranteed to be executed with no impact from the others. This decreases the risk of unexpected behaviour.

Another advantage of this approach is the possibility to use additional interfaces at a little cost. For instance, we have used a 3D navigator to basically get the information X, Y, Z, tilt, pan, roll that could also be easily obtained with a combination of 3 axis magnetometers, accelerometers and gyroscopes. As the interface with the 3D navigator is a set of bags receiving the position information, we can replace the 3D navigator with another device based on magnetometers, accelerometers and gyroscopes without changing the coordination rules defining the application.

VII. Conclusion

This paper has presented an innovative hardware and several frameworks easing the development of applications. The hardware, is a table combining a full HD display and a set of 384 RFID readers allowing to return the location of several tens of object tagged with RFIDs.

The frameworks are built on top of our in house rule-based middleware LINC. LINC relies on bags containing resources modelling our system, production rules and distributed transactions. A framework has been defined to map events coming from the table into resources stored in bags allowing the resources to be accessible with simple rules. This allows to react to event composing several RFID tags and to embed events verification in distributed transactions.

In addition, we have defined frameworks to offer a visual feedback to the user via displays. This includes 2D and 3D graphical objects rendering and multimedia contents such as videos.

This paper has shown firstly in isolation, through simple examples the genericity of these frameworks. Then, we have described how they may be combined and specialised via coordination rules to target two different application domains. This shows how the combination of the hardware, the middleware and the high level frameworks helps designing applications while offering a high degree of re usability of the frameworks components. The amount of work is decreased since a large part of the application is already available in the existing frameworks.

Future work is to enrich the tool kit around this table. We can integrate more external devices, sensors and actuators. For instance, cameras to deduce the number of people around the table (e.g., counting the detected faces), sensors to define the distance of the users from the table. We can also add voice interface. All these additional informations combined with the ones returned by the table may offer a richer user experience in order to target other application domains.

ACKNOWLEDGMENT

This work has been partially funded by the FP7 SCUBA project under grant nb 288079 and FUI Rapsodie project under grant nb F1209039V.

REFERENCES

[1] M. Louvel and F. Pacull, "A coordinated matrix of RFID readers as interactions input," in *SENSORDEVICES 2013, The Fourth International Conference on Sensor Device Technologies and Applications*, 2013, pp. 91–96.

[2] M. Louvel and F. Pacull, "LINC: A compact yet powerful coordination environment," in *Coordination Models and Languages*, ser. Lecture Notes in Computer Science, E. Kuhn and R. Pugliese, Eds. Springer Berlin Heidelberg, 2014, pp. 83–98.

[3] L.-F. Ducreux, C. Guyon-Gardeux, S. Lesecq, F. Pacull, and S. R. Thior, "Resource-based middleware in the context of heterogeneous building automation systems," in *IECON 2012, The 38th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2012, pp. 4847–4852.

[4] J.-M. Andreoli, F. Pacull, D. Pagani, and R. Pareschi, "Multiparty negotiation of dynamic distributed object services," *Journal of Science of Computer Programming*, vol. 31, pp. 179–203, 1998.

[5] D. Arregui, C. Fernström, F. Pacull, G. Rondeau, and J. Willamowski, "STITCH: Middleware for ubiquitous applications," in *sOc 2003, The second International Smart Object Conference*, 2003.

[6] N. Carriero and D. Gelernter, "Linda in context," *Commun. ACM*, vol. 32, pp. 444–458, April 1989.

[7] T. A. Cooper and N. Wogrin, *Rule Based Programming with OPS5*. Morgan Kaufmann, July 1988.

[8] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Boston, MA, USA: Addison-Wesley Longman Publishing, 1987.

[9] R. Wang and X. Qian, *OpenSceneGraph 3.0: Beginner's Guide*. Packt Publishing, 2010.

[10] R. Wang and X. Qian, *OpenSceneGraph 3 Cookbook*. Packt Publishing, 2012.

[11] F. Kerger, *OGRE 3D 1.7 Beginner's Guide*. Packt Publishing, 2010.

[12] M. Woo, J. Neider, T. Davis, and D. Shreiner, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[13] J. C. Mong and D. F. Brailsford, "Using SVG as the rendering model for structured and graphically complex web material," in *DocEng 2003, The 2003 ACM symposium on Document engineering*, New York, NY, USA: ACM, 2003, pp. 88–91. [Online]. Available: http://doi.acm.org/10.1145/958220.958236

[14] "HTML5," http://www.w3.org/html/wg/drafts/html/CR/.

[15] J. J. Garrett, "Ajax: A new approach to web applications," http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications, 2005.

[16] T. Brown, *Dynamic apache with ajax and json*, 1st ed. O'Reilly, 2006.

[17] F. Pacull, L.-F. Ducreux, S. Thior, H. Moner, D. Pusceddu, O. Yaakoubi, C. Guyon-Gardeux, S. Fedor, S. Lesecq, M. Boubekeur, and D. Pesch, "Self-organisation for building automation systems: Middleware LINC as an integration tool," in *IECON 2013, The 39th Annual Conference of the IEEE Industrial Electronics Society*, Nov 2013, pp. 7726–7732.

[18] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[19] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner, "Mires: a publish/subscribe middleware for sensor networks," *Personal and Ubiquitous Computing*, vol. 10, no. 1, pp. 37–44, 2006.

[20] J. Barbosa, F. Dillenburg, G. Lermen, A. Garzão, C. Costa, and J. Rosa, "Towards a programming model for context-aware applications," *Computer Languages, Systems & Structures*, vol. 38, no. 3, pp. 199–213, 2012.

[21] C. Julien and G.-C. Roman, "Egospaces: Facilitating rapid development of context-aware mobile applications," *IEEE Transactions on Software Engineering*, vol. 32, no. 5, pp. 281–298, 2006.

[22] M. Viroli, M. Casadei, S. Montagna, and F. Zambonelli, "Spatial coordination of pervasive services through chemical-inspired tuple spaces," *ACM Trans. Auton. Adapt. Syst.*, vol. 6, no. 2, pp. 14:1–14:24, Jun. 2011. [Online]. Available: http://doi.acm.org/10.1145/1968513.1968517

[23] M. Viroli, D. Pianini, and J. Beal, "Linda in space-time: an adaptive coordination model for mobile ad-hoc environments," in *Coordination Models and Languages*. Springer, 2012, pp. 212–229.

[24] M. Lutz, *Programming Python*. O'Reilly Media, Inc., 2006.