




Authoring AR Interaction by AR

Flavien Lécuyer¹ , Valérie Gouranton¹ , Adrien Reuzeau¹, Ronan Gaugne²  and Bruno Arnaldi¹

¹Univ Rennes, INSA Rennes, Inria, CNRS, IRISA

²Univ Rennes, Inria, CNRS, IRISA



Figure 1: From left to right, the user (a) adds an interactive behaviour on a bottle of glue, (b) imports the interactions in the environment, and (c) uses the interaction to pour the virtual glue from the real bottle into a virtual pot

Abstract

The demand for augmented reality applications is rapidly growing. In many domains, we observe a new interest for this technology, stressing the need for more efficient ways of producing augmented content. Similarly to virtual reality, interactive objects in augmented reality are a powerful means to improve the experience. While it is now well democratized for virtual reality, interactivity is still finding its way into augmented reality.

To open the way to this interactive augmented reality, we designed a new methodology for the management of the interactions in augmented reality, supported by an authoring tool for the use by designers and domain experts. This tool makes the production of interactive augmented content faster, while being scalable to the needs of each application. Usually in the creation of applications, a large amount of time is spent through discussions between the designer (or the domain expert), carrying the needs of the application, and the developer, holding the knowledge to create it. Thanks to our tool, we reduce this time by allowing the designer to create an interactive application, without having to write a single line of code.

CCS Concepts

• **Human-centered computing** → **Mixed / augmented reality**; Activity centered design; Interface design prototyping; Interaction design theory, concepts and paradigms;

1. Introduction

Augmented reality usage is rapidly growing in multiple domains [BCL15]. As the general demand for augmented reality experiences grows fast, the question of making the applications more interactive becomes more important too. While the design of interactive objects in virtual reality is now quite democratized, it is still difficult to provide such interaction in augmented reality. In the creation of applications, one of the main time-consuming tasks is to take into account the needs of the application. This requires many discussions between the client – who is often an expert in the concerned domain (e.g. an educational expert for training systems), and is not familiar with computer science – and the developer. The developer

will then write the code for the application, which also consumes a great amount of time. Often, even more time is necessary, because there is not only one developer, but a person in charge of writing the client’s needs, and a team of developers to produce the content. There is also a designer, between the client and the developer, who manages the conception of the application.

We propose to let the user create his own interactive augmented reality application, with an authoring tool. In the remainder of this paper, we will use the term *designer* for the one creating the application with the tool, regardless of their actual role (domain expert, designer, client, ...) Since there is a wide panel of applicative domains for augmented reality [Azu97], our tool is designed

to adapt easily to different contexts. To make the designer capable of designing interactive augmented reality applications, the tool is based on the use of extensible libraries. The interactions are created with a novel paradigm, based on existing work from virtual reality, which we adapted to take into account the difference between real and virtual objects, which is crucial for augmented reality. Those libraries allow a developer to create new interactions with an ensured reusability thanks to the object-relation paradigm [LC07]. Since the tracking is another difficulty for a novice, our tool integrates the management for the tracking through off-the-shelf solutions, making the overlay of virtual 3D objects easy to edit for the author of an application. The architecture of the tool allows the integration of other tracking solutions, so that a developer can enrich it and customize the tool to adapt it to the user's needs.

In this paper, after the analysis of related works in Section 2, we describe the problematic of authoring interaction for augmented reality in Section 3. We then give a brief overview of our solution in Section 4, to give more details in Section 5. We also illustrate the use of this solution with an example in Section 6. A video of this example is also available at <https://vimeo.com/349891514>.

2. Related works

In this section, we give an overview of works related to the authoring of interactive reality applications. For the creation of augmented reality applications, the designers need easy to use authoring tools, that allow to manage the placement of virtual objects according to the real world. For this, there are authoring tools designed to let a designer create an application with no code, albeit with little to no interaction. We start this section by presenting those works. To make interactive augmented reality applications, it is important to have interactions that are easy to implement, instead of having to code them anew for each application. To improve the creation of augmented reality applications, some works focused on easing the design of interaction for augmented reality. We describe them in the second part of this section. Another way of easing the creation of interactive applications is to improve the reusability of the code. This has been done for virtual reality, with the object-relation concept, which we focus on in the last part of this section.

As augmented reality is a strong tool for the presentation of augmented information, some tools aim at providing the means for an author to create augmented presentations. The first of them, PowerSpace, is based on the use of the slideshow editor PowerPoint, which is well-known by the general user, and allows them to be more efficient during the authoring [HR02]. However, the content authored with PowerSpace is limited to the capacity of a slideshow. This slideshow format can also be authored directly in-situ, as done with ASMIM [NMTS18], where the author can create steps based on the environment, augmented with some layers of information. A similar effort can be found with AMIRE-ES [HSZ05], where the author can create an augmented reality guide that can be used with different devices. In ACARS, the authors also propose a limited customization for some of the properties of the objects according to the user's preferences [ZON13]. The combination of on-site and off-site use allows a more refined authoring. This in-situ authoring can also be adapted for the augmentation of outdoor environments,

as with Sketching up the world [LMZ*12]. A more immersive presentation of the content can be found by using outdoor AR, where the augmentation is provided at given points in space. To this extent, Guven et al propose to augment a 3D space by selecting specific points, and associating content to it [GF03]. Similarly, AR Game Builder associates content to GPS positions [KS10]. This content takes the form of text and images to narrate a story. Finally, AREDA proposes to record directly the user to generate an application [BW19]. The application generated is similar to a slideshow, with each step displayed when the object placement from the previous step is done by the user.

Augmented reality can be used with more complex interactions. In Virtual Museums, the authors propose to associate 3D models with fiducial markers [LSB*04]. The augmentation is interactive and reacts to the state of the markers, and can be customized through a desktop interface to modify the visualisation of the object (e.g. color, position, ...). The authoring can also be done in immersion in the environment. By providing a tangible interface for the authoring, Lee et al. allow the user to act directly on the environment with simple metaphors [LNBK04], with some simple interactions for the augmented objects. For more customizable interactions, Broll et al. propose to add behaviours on the objects [BHB08]. The behaviours come from scripts and are attached automatically on certain objects, making it difficultly usable for a domain expert. The same observation can be made for AR-Room, where the behaviours must be created with scripts by a computer scientist [Par11]. This system of behaviours is also present in APRIL to make the objects interact, although the interactions are still strongly oriented towards augmented presentations [LS05]. Those behaviours can be triggered by certain conditions, such as the contexts presented in k-MART [CKL*10], in which the user can select when a given behaviour is to be used. It is to note, however, that the behaviours proposed are quite limited. In DART, the behaviours function as a combination between cues and actions, which make an object react to certain conditions (e.g. occlusion of a marker) with a given action (e.g. an animation) [MGDB04]. The user can combine both as they want to generate more complex behaviours, but the available cues are limited to what a marker can do. A similar proposition is made with BEAR, where a given set of states can be associated with actions on the objects [LC10]. However, the creation of the behaviours require some knowledge in computer science, making the tool less adapted to a designer.

In virtual reality, more attention have been put on the efficient design of interactions. Particularly, the object-relation paradigm aims at improving the reusability of the interactions that have been defined, through the separation of the action (represented by the relation) and the objects involved. Although any number of object can be involved in a relation, this is particularly interesting for actions that require more than two objects, as the complexity remains low on the developer's end. It has first been proposed with the model *Cause & Effects* [LC07]. It has also been used by the models *Storm* [MGA07], and *Mascaret* [CTB*12], which integrated a graphic model to simplify the use of their models. Lanquepin et al. also proposed Domain-DL, integrated in the platform *HUMANS* [LCL*13], to benefit from the other models in the platform, such as a scenario model. The most recent model is #FIVE [BGB*15],

which proposes a framework independent from any platform, to make it more versatile.

To conclude this discussion about the existing authoring tools for AR, an important observation can be made: for now, the authoring tools have mostly followed the tendency of using AR mainly as a presentation means. Although some other tools tried to bring more interaction into AR, their lack of affordability prevented them from truly attaining this goal. In parallel, some models have emerged for virtual reality to manage this kind of interaction, but they only manage virtual objects, and cannot be used as is for AR.

3. Needs

Few augmented reality applications have put a focus on the interaction. In order to democratize interaction for augmented reality applications, we need to provide efficient ways to create it. In most works, the position and occlusion state of the real objects are observed, and used to trigger some behaviours - usually, some animation - once certain conditions on those properties are verified. Because of this, the interactions between the real and virtual objects are often limited (mostly occlusion and collisions). However, a higher level of interaction could be proposed in augmented reality, similar to the one provided in virtual reality, where the objects can have complex behaviours and evolve according to the state of the world and the user's actions.

Indeed, the manipulation of real objects is a powerful means to make the user get a feeling akin to the reality, while the overlay of virtual elements gives more information and improves the experience. Thanks to this, and although augmented reality is more constrained than virtual reality, it has a great potential in terms of fully interactive experiences. Thanks to many works on it, the tracking solutions for augmented reality can now be considered sufficient for the average needs, at least for indoor applications [MUS16]. However, interactions for augmented reality have been far less studied. Because of this, it is still usually the role of the developer to create such interactions, according to the needs of a designer.

4. Solution overview

To support the creation of such interactive augmented experiences, we propose an authoring tool, using a novel methodology for the implementation of the interactions, based on the object-relation paradigm already used in virtual reality. To palliate the lack of interactivity in augmented reality applications, we propose a new framework for the creation of interactions in augmented reality. This framework is divided in two parts: on the side of the developer, we provide a new methodology to design and implement the interactions. For the designer – whom can be a domain expert, an artist, or a graphist for instance – we provide an authoring tool based on this novel methodology, to make an easy use of the Both parts are designed in a scalable way, in order to make it easy to adapt to any domain of application.

Usually, the work of the developer consists in fulfilling the demands of the designer, with the difficulty induced by the difference in their vocabularies. With our framework, the role of the developer is deported upstream. Indeed, the developer becomes in charge of

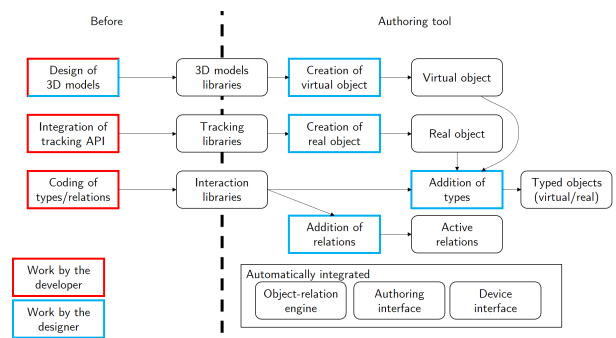


Figure 2: The main concepts of the framework

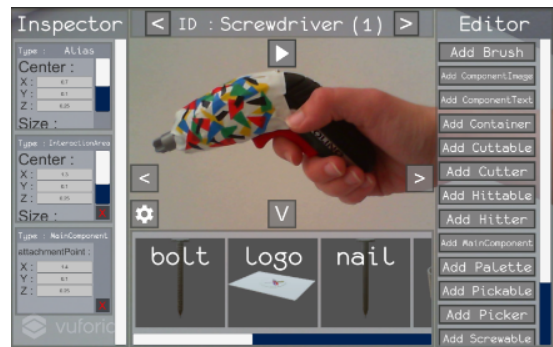


Figure 3: The inspector for the types in the authoring tool

the creation and the completion of the libraries used by the authoring tool, which can then be used by the designer. The use of those libraries also facilitates the use of the tool for a new domain of application. To ensure an easy reuse of the interactions, as well as a good efficiency during the design of those interactions, we designed a new methodology for the design and implementation of the interactions, based on the object-relation paradigm. More details about this are given in Section 5.1. In the same way, the tracking is managed as the integration of third party software. The integration is managed in a modular way, so that any third party software can be used, according to what is needed for the application.

On the other side, the designer is given an authoring tool, which is enriched by those libraries, to create interactive augmented reality applications. The main interface of this tool is illustrated in the Figure 3. As illustrated in Figure 2, those libraries are then used to enrich the authoring tool, which can in turn be used by the designer to create an interactive augmented reality application.

To help create interactive applications with this novel paradigm for augmented reality, we provide an authoring tool based on it, to easily create interactive augmented reality applications. In the rest of this paper, we present more details in Section 5, which we illustrate with an example in Section 6.

5. Authoring AR by AR

In this section, we give more details on the methodology and authoring tool to create interactive augmented reality applications.

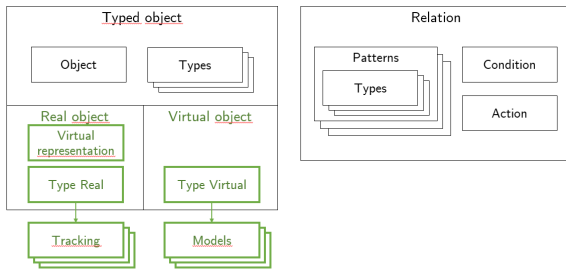


Figure 4: To extend the object-relation to AR, we make the distinction between real and virtual objects, with links to the tracking and 3D models libraries respectively

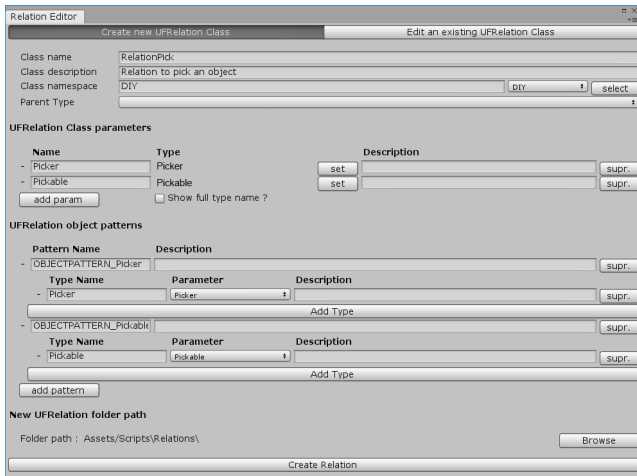


Figure 5: An editor is provided to ease the creation of new relations

First, we describe how the interactions are designed in Section 5.1, and how they are used in the authoring in Section 5.2. We then present how the tracking and virtual objects are managed in the authoring, in Sections 5.3 and 5.4 respectively. Finally, we discuss how the authoring data is managed in the tool in Section 5.5.

5.1. Creation of the interactions

The interactions are created by the developer, with the help of the object-relation paradigm. The object-relation paradigm is a paradigm created to manage interactions for virtual reality. Its goal is to design the interactions in such a way that encourages the reuse, instead of the reimplementing.

Usually, the object-relation models are based on two main concepts: the types objects and the relations (see Figure 4). The objects in the environment are made interactive by receiving types (e.g. *Screw*), which gives them behaviours (e.g. to turn when used), and bear properties that can be used by the relations to interact with the objects (e.g. a screw type). The types can also be defined parametric types, using values to modify their behaviour. For instance, with a single type *Screwable* and a value *screw_type*, multiple screw types can be easily defined. The types can also be enriched with variables representing the current status of the object.

Those variables can be manipulated by the relations to modify the behaviour and status of the objects. In complement, the relations define the needed types for the objects, a condition for the execution (e.g. compatible screw types), and an action to execute (e.g. screw and/or display some text). The object-relation paradigm also allows to create relations involving more than two objects, such as soldering. For this, a relation with four objects (a soldering iron, two pieces to connect, and a metal source) could be created. This separation between the types, representing the capacities of the objects, and the relations representing the actions, makes the definition of the interactions more flexible.

For the developer, the interest of this approach is twofold. First, the use of the object-relation paradigm allows to gain time during the coding of interactions, whether they are reused or implemented using the interface. Secondly, the work of the developer is pushed upstream in the development process. This is particularly useful, as the work of the developer can be retargeted on the creation of code libraries, instead of being constantly going and coming with the designer to adapt the code to their needs.

For the use with augmented reality, we add a particular type on the objects, depending on whether it is virtual or physically real. This allows to define different behaviours for a same interaction, depending on the nature of the objects interacting. For instance, assembling a virtual tip on a real screwdriver will not be managed the same way as if the tip was real and the screwdriver virtual.

To make the creation of new interactions easier and faster, an C# interface is provided to create new relations. A graphical editor is also integrated in Unity, for both the types and relations, and can be used to ensure that the relation is properly built. An example, for the relation *RelationPick*, is given in Figure 5.

5.2. Integration of the interactions in the authoring tool

In the authoring tool, the types and relations defined in the libraries by the developer can be used as add-ons by the designer.

As can be seen in Figure 3, when an object is detected by the tracking system, a panel with its current types will be automatically displayed on the left of the screen, and a second panel with the available types is displayed on the right. This allows the designer to add and customize as many types as needed for a single object. For the screwdriver on Figure 3, we can see the type *Alias* that is already attached on the object, as well as the *InteractionArea* one, that can be modified to customize the bounding box that will be taken into account for the interactions. The *MainComponent* one is used to let the screwdriver interact with the tips.

The types that are mandatory for the object, such as the *Alias* or *Virtual* one (depending on the physical/virtual nature of the object), are placed on the object from the start and cannot be removed by the designer. As such, they are not proposed on the available types, and are locked on the inspection of current types. Similarly to the list of available types, a list of available relations can be accessed from the authoring tool. On the example in Figure 6, no relation are used in the application. However, some relations on the DIY theme are proposed and can be selected by the designer. For the screwing action, for instance, the designer would need the

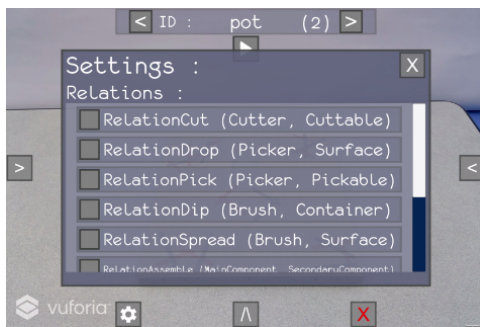


Figure 6: The list of relations in the tool

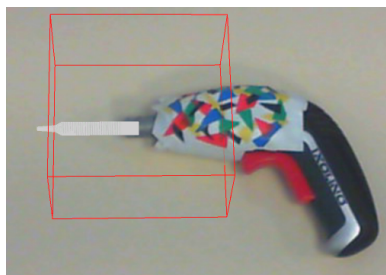


Figure 7: A real screwdriver, with its interactive area in red. The virtual representation is voluntarily modified to change the interactive zone

relations `RelationAssemble` and `RelationScrew` that are proposed (respectively to assemble the tip on the base, and to screw with the screwdriver tip).

This integration of the types and relations is especially appreciable for the designer, as it allows them to experiment with the available interactions by themselves, without having to request the help of a developer to do so. Since the integration of the types and relations in the authoring tool is automatic, the work of the developer is also eased here.

5.3. Integration of the tracking libraries

To prevent any constraint from the choice of a single tracking API that becomes integrated in the authoring tool, we decided to integrate the tracking third parties as modules that can be added to the application. Thanks to this, the choice in the tracking API can be made according to the needs of the designer. We defined a minimal common API to communicate with any tracking system, in order to ensure its genericity. To make the rest of the application independent from the tracking software, each real object is represented virtually in the environment. This virtual representation has the `Alias` type affixed to it. The `Alias` type is in charge of updating the virtual representation's pose according to the tracking information, and can be used by the relations to get information on the object, or for occlusion for instance. Although any kind of object can be tracked, given that the tracking software in use is able to detect it correctly, the use of a complex model makes it rapidly quite heavy to compute, and difficult to manipulate for the designer.



Figure 8: List of virtual objects as it appears in the application

To prevent this, the virtual representation and interactive areas are automatically simplified to rectangular bounding boxes, as shown in Figure 7. As an example, we already integrated Vuforia using this architecture. For the case of Vuforia, the virtual representations are set as Vuforia targets, which are then automatically updated by the tracking API. Those representations also bear the `Alias` type, which will be used by the relations to get the necessary information about the real object. For the developer, this just means that the third party API needs to be integrated normally, with the addition of the `Alias` type on the virtual representation used to translate the tracking information in the virtual environment. The real objects are then registered in a library, which is used by the authoring tool to automatically recognize them.

5.4. Integrate the 3D models

Since both the developer and the designer can bring their own 3D models, it is important to let them both have a way to integrate them in the application. The 3D models provided for the creation of virtual objects in the authoring tool are retrieved from a specific folder on the device on which the authoring tool is installed. To provide the designer some basic virtual objects to use in the authoring tool, some simple 3D models are provided by default. Others 3D models can be added by placing the files in the specific folder. However, to ease the process, the designer can import new 3D models directly from the authoring tool. This lets the designer modify by themselves the contents of the library, with their own 3D models if needed. Thanks to this, a developer is no longer required to integrate the 3D models in the application. As the authoring tool is implemented using the Unity game engine, the format for the 3D models are the ones supported by Unity (OBJ, FBX, etc.). Once those objects are added to the authoring tool, they will then appear in a list, as displayed in Figure 8. This list can then be used by the designer to add objects in the scene.

5.5. Export of the content

To use the application after the authoring is done, the designer needs a way to export the created content. Inside of the authoring tool, the data is structured in three parts, as shown in Figure 9: the scene, the interactions, and the tracking information. Those three parts are integrated in a Unity project that is used in real time by

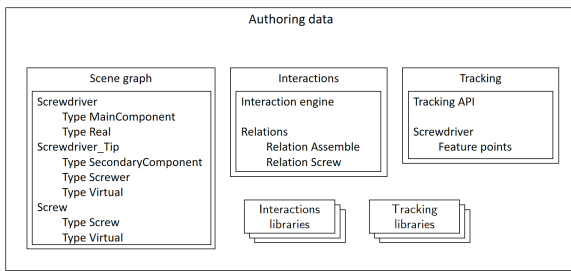


Figure 9: The export from the authoring tool generates the code from the application

the authoring tool, and that can also be exported for further use, such as refining the animations for a better experience. The choice of those three parts is motivated by the wish to keep the libraries as well separated as possible from the content that has been authored, which in turn makes it simpler for a developer to reuse the generated code. To do so, the content is integrated in the scene, and references the different libraries. As illustrated in Figure 9, the authoring generates the scene, in which the objects are enhanced with types. Some specific Unity objects are also added for the management of the tracking and the relations. First, the scene created from the authoring tool is exported. Since the virtual objects are managed by a scene graph, in which each object is placed relatively to another one that already exists, the scene in the Unity project is constructed with this same scene graph.

The code for the interactions is distributed between the two main entities of the object-relation paradigm: the relations, and the types. The types are added as Unity components on the objects, that can easily be removed or added from the Unity editor after the authoring. The relations are added in the Unity project as separate objects, with a behaviour coming from the relation itself. The types and relations are created using the framework #FIVE [BGB*15], which we extended to manage augmented reality. This choice is motivated by the strong independance of #FIVE regarding the target device, which makes it easier to adapt for augmented reality. The engine that manages the test and launch of the relations is also using #FIVE, and is automatically included in the project. Since the relations and types are taken from libraries, there are not supposed to be modified by a developer. However, this is still possible.

Thirdly, the tracking information is integrated in the Unity project, using the format preferred by the tracking API in use. Usually, the tracking information is put on each virtual representation, so that it can be tracked when the project is launched.

The use of multiple third parties, such as #FIVE to manage the relations, and the use of third party tracking, is motivated by the need for the tool to be as scalable as possible. To this end, the tool is also designed to be simple to adapt to other third parties.

6. Example of use

In this section, we detail how the authoring tool can be used, through an example. First, we present a use case in Section 6.1. We chose the domain of DIY for this use case as the actions for this

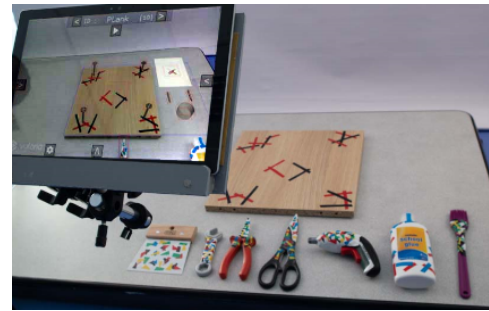


Figure 10: Setting of the application, with the tablet and the real objects

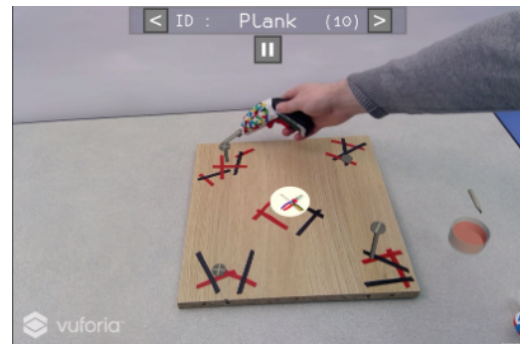


Figure 11: Example of an interaction from the use case. The user is using the screwdriver to drive the top left screw

domain are quite well-known, but the approach could be, of course, scaled to other domains. Then, we illustrate how a designer would add a real object for such a use case.

6.1. Use case

In order to demonstrate the use of our authoring tool, we designed a short use case for illustration sake. The goal of this use case is to build an application in which a user will build a piece of furniture with different tools. The application, designed specifically to demonstrate the authoring, uses several real props, as well as a tablet to augment the environment, as shown in Figure 10. The realization of the piece of furniture is divided in two tasks: an assembly and a screwing task. The user can do those two tasks in any order.

The first task consists in placing a virtual logo on the real plank. To do it, the user must first take some virtual glue with a real brush and spread it on the plank. Then, the user must prepare the logo by cutting it out from a virtual piece of paper, and place it on the plank, where the glue is. Finally, the task is completed by using a real pallet to flatten the logo on the plank.

The screwing task consists in screwing some virtual screws and bolts on a real plank. In this task, there are three virtual screws: two cross-head ones and a slot-head one, as well as a virtual bolt to tighten. To help in the task, the user can use an electric screwdriver – a real one – with different virtual tips, and a real wrench. The use of the screwdriver is illustrated in Figure 11.

With these objects and interactions, a complete interactive application can be generated without the need for the author to know anything about coding. For someone who is not yet familiar with the tool, the creation of this application can be done in around an hour and a half, considering that the developer has already configured the tracking system. In this time, five minutes are used to manage the relations, and approximately five minutes are spent for each object, to select the types and configure them. The work by the developer prior to the authoring is difficult to estimate, as it depends highly on the tracking system to integrate.

In order to illustrate how the designer would use the tool for the creation of an application, let us study a short example. In this example, and unless specified, the designer is the one doing the authoring. In many cases during the use of an augmented reality application, a real object is used to affect the virtual environment. To create an interactive environment, the designer can add virtual objects, real objects and interactions, in any order.

6.2. Setting of a real object

For the addition of a real object, the designer needs to initialize the tracking and configure the real object in the environment. It is to note that the setting of a virtual object is mostly the same, except that the first three steps are replaced by the selection of a 3D model in the tool. This is done through the following steps:

Tracking initialization First, the designer must get tracking information about the object. For this, the designer can scan the object beforehand, to get the tracking features that will serve for the recognition afterwards. With the tracking system we used for this example, Vuforia, the designer can place the object on a reference pattern, use the provided application (*Scanner*) to get the tracking data and use it in the authoring tool. This is done easily, by adding a Vuforia object recognition manager for the real object, directly in the game engine by using a prefab and selecting the file containing the features for the intended object. This prefab is integrated in the authoring tool, and uses the scripts from the tracking API to make them easier to use. Although this can be done by a designer, this is still easier for a developer. Of course, this tracking initialization process is dependent of the tracking system in use, so this step can differ with another system.

Creation of the virtual representation Once the object is recognized in the environment, a virtual representation of it is automatically created and co-localized with the real object. Once the virtual representation is added, it is automatically used to generate occlusion with the virtual objects. To make it simpler for the author, the virtual representation is not displayed. This lets the designer act on the real object in a more transparent manner.

Addition of the Alias type Once the object is recognized and its appearance is set; the `Alias` type is automatically added on it to state that the object with this type is in fact the virtual representation of an object tracked in the real world. This type modifies the activity of the object according to this fact.

Addition of the types for the object When the object is integrated in the environment, it is time for the designer to define what types it should bear. This can be done using the object inspector. As

in Figure 12, the types that can be added are displayed on the right panel. The ones already present on an object are displayed on the left panel, from which the designer can customize their properties. In this inspector panel, the designer can get the information about the object and its types. If the type bears some properties – as, in this case, a list of possible contents – the value can be modified from the inspector too.

For the objects to interact with each other, the designer chooses the relations to use as being possible relations for the environment. To do this, a list of available relations is proposed to the author, as shown in Figure 6. In this list, the designer can select the relation they want and add it to the scene. Once the relation is added in the scene, it can be used by the object-relation engine, which will try to match objects to launch the relations.

7. Conclusion

In this paper, we presented a new methodology for the creation of interactions in augmented reality, supported by an authoring tool. Our goal with this methodology and tool is to give to the designers of all domains the opportunity to create interactive experiences with augmented reality, without feeling limited by the technology barrier. Giving the designers this opportunity is particularly important, as the democratisation of augmented reality can difficultly be reached without an important contribution from the designers.

To make the interactions easily accessible for those designers, we designed a specific paradigm for the coding of said interactions. This paradigm is based on the object-relation paradigm, which is quite useful for the creation of interactions in virtual reality. We adapted this paradigm for augmented reality, by letting it take into account whether an object is real and virtual. With this paradigm, the interactions created by a developer are by nature easy to reuse, and will enrich libraries for the authoring tool.

Thanks to this novel paradigm for the design of interactions, and to the authoring tool supporting it, the creation of interactive augmented reality experiences is made easier for people without particular knowledge in computer science. This paradigm and this tool

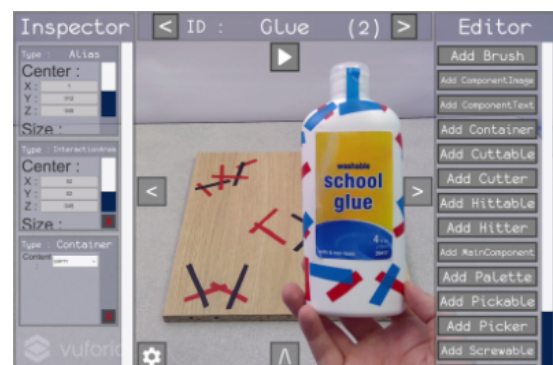


Figure 12: Inspector for an object. The left panel indicates the current types for the object, and the right one proposes the available types to add on the object.

can also be used directly by computer scientists, to help them design their applications faster. The sources of the authored application can also be retrieved, to be modified externally by a developer who would want it.

To be able to evaluate further our solution, we would like to use it on other domains. This would allow to create new libraries for other domains, to enrich the list of libraries to include in the authoring tool. For instance, we think about using this authoring methodology for the medical domain, which presents many different objects, each with a precise use. Because of this, it would allow us to work on a consequent set of new relations, types, and 3D models.

In the same kind of idea, it would be interesting to integrate new tracking libraries. For instance, since many applications use fiducial markers as their tracking targets, ARToolkit would be a good addition to the authoring tool [BPKM00].

Acknowledgements

This work is part of the ANR-16-FRQC-0004 INTROSPECT project, and the SUNSET project funded by the ANR-10-LABX-07-01 “Investing for the Future” program.

References

- [Azu97] AZUMA R. T.: A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments* 6, 4 (8 1997), 355–385. 1
- [BCL15] BILLINGHURST M., CLARK A., LEE G.: A Survey of Augmented Reality. *Foundations and Trends in Human-Computer Interaction* 8, 2-3 (3 2015), 73–272. 1
- [BGB*15] BOUVILLE R., GOURANTON V., BOGGINI T., NOUVIALE F., ARNALDI B.: #FIVE : High-Level Components for Developing Collaborative and Interactive Virtual Environments. In *Proceedings of Eighth Workshop on Software Engineering and Architectures for Real-time Interactive Systems (SEARIS 2015), conjunction with IEEE Virtual Reality (VR)* (Arles, France, Mar. 2015). 2, 6
- [BHB08] BROLL W., HERLING J., BLUM L.: Interactive Bits: Prototyping of Mixed Reality Applications and Interaction Techniques through Visual Programming. In *2008 IEEE Symposium on 3D User Interfaces* (Mar. 2008), pp. 109–115. 2
- [BPKM00] BILLINGHURST M., POUPYREV I., KATO H., MAY R.: Mixing realities in Shared Space: an augmented reality interface for collaborative computing. In *2000 IEEE International Conference on Multimedia and Expo. ICME2000. Proceedings. Latest Advances in the Fast Changing World of Multimedia (Cat. No.00TH8532)* (7 2000), vol. 3, pp. 1641–1644 vol.3. 8
- [BW19] BHATTACHARYA B., WINER E. H.: Augmented reality via expert demonstration authoring (AREDA). *Computers in Industry* 105 (Feb. 2019), 61–79. 2
- [CKL*10] CHOI J., KIM Y., LEE M., KIM G. J., NAM Y., KWON Y.: k-MART: Authoring tool for mixed reality contents. In *2010 IEEE International Symposium on Mixed and Augmented Reality* (Oct 2010), pp. 219–220. 2
- [CTB*12] CHEVAILLIER P., TRINH T.-H., BARANGE M., DE LOOR P., DEVILLERS F., SOLER J., QUERREC R.: Semantic modeling of Virtual Environments using MASCARET. In *2012 5th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)* (3 2012), pp. 1–8. 2
- [GF03] GÜVEN S., FEINER S.: Authoring 3D hypermedia for wearable augmented and virtual reality. In *Seventh IEEE International Symposium on Wearable Computers, 2003. Proceedings.* (Oct 2003), pp. 118–126. 2
- [HR02] HARINGER M., REGENBRECHT H. T.: A pragmatic approach to augmented reality authoring. In *Proceedings. International Symposium on Mixed and Augmented Reality* (Oct 2002), pp. 237–245. 2
- [HSZ05] HALLER M., STAUDER E., ZAUNER J.: AMIRE-ES: Authoring Mixed Reality once, run it anywhere. In *Proceedings of the 11th International Conference on Human-Computer Interaction (HCI)* (2005), vol. 2005. 2
- [KS10] KLOPPER E., SHELDON J.: Augmenting your own reality: Student authoring of science-based augmented reality games. *New Directions for Youth Development* 2010, 128 (2010), 85–94. 2
- [LC07] LUGRIN J.-L., CAVAZZA M.: Making sense of virtual environments: action representation, grounding and common sense. In *Proceedings of the 12th international conference on Intelligent user interfaces* (2007), ACM, pp. 225–234. 2
- [LC10] LUGRIN J.-L., CAVAZZA M.: Towards AR game engines. In *3rd workshop on software engineering and architecture of realtime interactive systems SEARIS* (2010), Latoschik M. E., Reiners D., Blach R., Figueroa P., Dachselt R., (Eds.), Shaker Verlag. 2
- [LCL*13] LANQUEPIN V., CARPENTIER K., LOURDEAUX D., LHOMET M., BAROT C., AMOKRANE K.: HUMANS: a Human Models based Artificial eNvironments software platform. In *VRIC'13* (Laval, France, 3 2013), pp. 59–68. 2
- [LMZ*12] LANGLOTZ T., MOOSLECHNER S., ZOLLMANN S., DEGENDOERFER C., REITMAYR G., SCHMALSTIEG D.: Sketching Up the World: In Situ Authoring for Mobile Augmented Reality. *Personal Ubiquitous Comput.* 16, 6 (Aug. 2012), 623–630. 2
- [LNBK04] LEE G. A., NELLES C., BILLINGHURST M., KIM G. J.: Immersive Authoring of Tangible Augmented Reality Applications. In *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality* (Washington, DC, USA, 2004), ISMAR '04, IEEE Computer Society, pp. 172–181. 2
- [LS05] LEDERMANN F., SCHMALSTIEG D.: APRIL: a high-level framework for creating augmented reality presentations. In *IEEE Proceedings. VR 2005. Virtual Reality, 2005.* (March 2005), pp. 187–194. 2
- [LSB*04] LIAROKAPIS F., SYLAIU S., BASU A., MOURKOUSSIS N., WHITE M., LISTER P. F.: An Interactive Visualisation Interface for Virtual Museums. In *VAST 2004: The 5th International Symposium on Virtual Reality, Archaeology and Cultural Heritage* (2004), Chrysanthou Y., Cain K., Silberman N., Niccolucci F., (Eds.), The Eurographics Association. 2
- [MGA07] MOLLET N., GERBAUD S., ARNALDI B.: Storm: a generic interaction and behavioral model for 3D objects and humanoids in a virtual environment. In *IPT-EGVE The 13th Eurographics Symposium on Virtual Environments* (2007), pp. 95–100. 2
- [MGDB04] MACINTYRE B., GANDY M., DOW S., BOLTER J. D.: DART: A Toolkit for Rapid Design Exploration of Augmented Reality Experiences. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2004), UIST '04, ACM, pp. 197–206. 2
- [MUS16] MARCHAND É., UCHIYAMA H., SPINDLER F.: Pose Estimation for Augmented Reality: A Hands-On Survey. *IEEE Transactions on Visualization and Computer Graphics* 22, 12 (12 2016), 2633–2651. 3
- [NMTS18] NGUYEN T. V., MIRZA B., TAN D., SEPULVEDA J.: AS-MIM: Augmented Reality Authoring System for Mobile Interactive Manuals. In *Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication* (New York, NY, USA, Jan. 2018), IMCOM '18, ACM, pp. 3:1–3:6. 2
- [Par11] PARK J.-S.: AR-Room: a rapid prototyping framework for augmented reality applications. *Multimedia Tools and Applications* 55, 3 (Dec 2011), 725–746. 2
- [ZON13] ZHU J., ONG S. K., NEE A. Y. C.: An authorable context-aware augmented reality system to assist the maintenance technicians. *The International Journal of Advanced Manufacturing Technology* 66, 9 (Jun 2013), 1699–1714. 2