Component-based middleware for distributed augmented reality applications

Mehdi Chouiten
IBISC Laboratory
40 Rue du pelvoux
91080 Courcouronnes, France
+(33) 1.69.47.06.18
Mehdi.Chouiten@ibisc.fr

Jean-Yves Didier
IBISC Laboratory
40 Rue du pelvoux
91080 Courcouronnes, France
+(33) 1.69.36.39.14
Jean-Yves.Didier@ibisc.fr

Malik Mallem
IBISC Laboratory
40 Rue du pelvoux
91080 Courcouronnes, France
+(33) 1.69.47.75.15
Malik.Mallem@ibisc.fr

ABSTRACT

This paper describes the design and implementation of a middleware for a framework dedicated to Augmented Reality / Mixed Reality (AR/MR) applications. The goal is to offer an environment for the development of distributed applications running on mobile devices (wearable computers and/or smartphones). The paper first presents the main needs of an AR application and introduces the necessity of distribution in this field. Then we make a quick overview of existing distributed AR frameworks. The goal of this overview is to extract main features and strengths of each framework's architecture based on a set of defined criteria.

This comparison being meant as a starting point to extend our own framework (ARCS: Augmented Reality Component System), the last section is about the design and implementation of our own software infrastructure for transparent distributed Augmented Reality taking in consideration our own goals and constraints and taking profit of the strengths of the studied existing frameworks.

Categories and Subject Descriptors

C.2.1 [Network Architecture and Design]: *Distributed networks*

H.5.1 [Multimedia Information Systems]: *Artificial, augmented, and virtual realities*

General Terms

Design.

Keywords

Augmented Reality framework; Middleware design and implementation; Component-based architecture; Distributed Systems; Mobile Devices.

1. INTRODUCTION

In brief terms, Augmented Reality is a growing field that aims to offer systems able to add virtual entities to a given real environment in real time. These entities (augmentations) are most of the time visual augmentations of the real scene by virtual

COMSWARE 2011, July 04-07, Verona, Italy

DOI 10.4108/comsware.2011.8

applications as a set of linked components communicating with each other (locally or through a network) which makes the code easier to maintain and increases the reusability possibilities.

Distributed applications in the AR context are used in order to enable users (at remote sites or using different terminals) to collaborate on a common task. In the same scope, distribution is also a mean to run AR applications on mobile terminals with limited computation power by using computation offloading techniques. Previous works on distributed architectures for simulations and Virtual Reality have already produced results. Even if these systems were not applied on AR, they partially inspired some of the current AR frameworks. Some of the most significant standards for simulation are ALSP (Aggregate Level Simulation Protocol [13]), DIS (Distributed Interactive Simulation [14]) and HLA (High Level Architecture [15]). As pointed out by a recent survey [17], constructing a pervasive middleware to support AR systems is still a challenge.

In this paper, we are going to review five of the most used distributed frameworks for Augmented Reality: DWARF, STUDIERSTUBE, MORGAN, VARU and TINMITH. Even if there are many more other frameworks for AR, the choice of these systems is due to the fact that they manage distributed applications and to their wide use which gives a significant feedback on these still maintained projects. We will explain the architectures main concepts and features and compare them before extracting main strengths of each. It is important to notice that these frameworks don't offer exactly the same features and thus, have different architectural constraints.

The strengths of each architecture being identified, we build our own architecture having in mind our own goals and constraints on one hand and the teachings from other architectures on the other hand. The last part is then the description of the architectural design and implementation choices within this context.

2. Frameworks Overview

2.1 DWARF

DWARF [1] is a component-based framework allowing rapid prototyping of AR applications. It uses the concept of interdependent distributed services which *needs*, *abilities* and *connectors* are exposed with the help of a *service manager*. A service offers one *ability* to other services and requests its *needs* from them (see Figure 1).

There is only one service manager per network node. Each of them controlling its local services and cooperating with the other

managers to connect to remote services. DWARF is decentralized; it doesn't need a central server to run applications. The most common services are already developed. The framework includes a task-flow engine (sequence of actions to be done by the user), a user interface engine, a tracking subsystem and a world model description system collecting several data on the system's user and its environment. Technically, DWARF distribution is based on CORBA middleware and CORBA IIOP protocol.

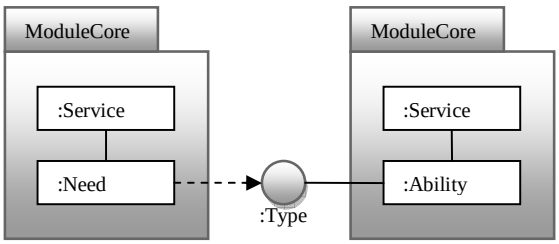

Figure 1. DWARF services connection

2.2 The MORGAN Framework

MORGAN [9] is a also component-based framework. It is convenient for Multi-User AR and VR projects. In the MORGAN paradigm, projects are composed of sets of components that subscribe to input devices (e.g. Tracking devices) they are interested in. These devices provide new samples at different rates. This is implemented using a *publisher-subscriber* pattern. The framework also offers a rendering engine specially designed for supporting multiusers within a distributed framework.

The creation, deletion and retrieval of components is done via a core component called *broker*. To implement the creation of remote components, the *factory method* pattern has been used by the MORGAN system developers. Like DWARF, MORGAN uses CORBA middleware but it also implements a proxy pattern for other protocols. Unlike DWARF, the MORGAN framework is not open source and there are no public tutorials available.

2.3 Studierstube

Here, each component is called an *application object*. This concept encloses the data, its graphical representation and the application operating on the data. Each application object inherits from an application type. Studierstube[5] allows instantiation of different application objects at the same time from the same or different application types. These applications can communicate to share features and data. Studierstube architecture is centralized since it requires a *session manager* as central server [16]. Apart from the interaction metaphor that we have tested, Studierstube applications written in Open Inventor (OIV) can be seen as distributed scene graphs replicated for each host. Distributed applications are based on Distributed OpenInventor (DIV) which is an extension of Open Inventor with the concept of distributed shared scene graphs.

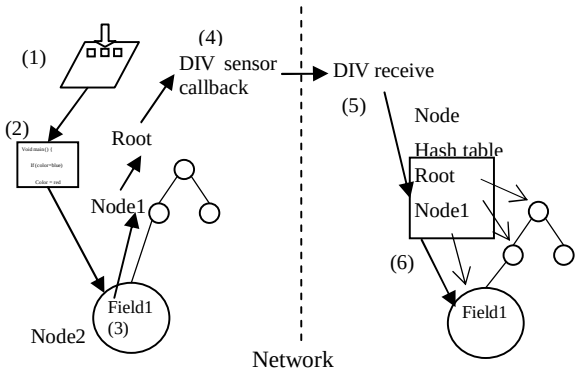

Figure 2. Studierstube event propagation over the network

To maintain the consistency of the graph over the network, modifications are propagated as shown in figure 2. First step (1) is the user pressing a button, (2) is the corresponding code execution, (3) is the modification of Field1 on Node2 locally. The modification is then propagated upwards to the root and notified to a sensor which transmits the message to the remote host(4). Finally, the receiver looks for the concerned node (5) and applies the changes (6).

2.4 Tinmith

The architecture of the framework (Tinmith-evo5 [11]) is modular to support a wide range of AR and multimedia applications. Some modules can be application specific (e.g. navigation module) when others are generic. The communication between modules is made possible by a client-server style architecture. A module providing data is the server that listens to clients that request a subscription. When the data on the server is changed, the new values are sent to all clients that have registered their interest to this message. The clients can then use the new data to perform the task of the module (e.g. refresh the display).

The system is asynchronous and data driven. If there is not any new data, no new message will be generated and no action performed by any software module.

Figure 3 shows how the Objects are connected. Objects in the system are in C++ native format. The Binary of the object is specific to the running process so it is not possible to send it directly across the network. Serialization not being available by default in C++, a custom serialization mechanism has been developed using an XML structured format.

When the source and the listener are both running on same CPU, they are connected via a callback function call used by the source to notify data changes to the listener.

When the connected Objects are on distant terminals, as in Figure 3(2), the Source's new data is serialized by Object Tx which transmits it over the the network. The Rx object receiving the new value deserializes it and signals to the listener the data modification. The middleware used is tinmith specific and the protocol can be TCP or UDP based (different implementations).

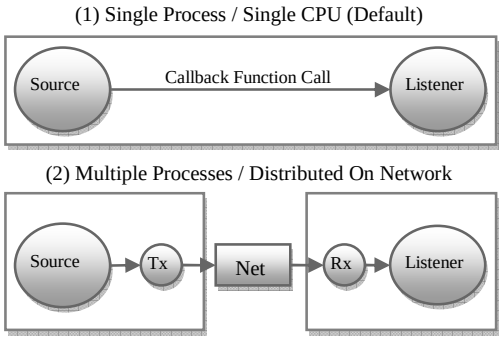

Figure 3. Tinmith transparent network distribution

2.5 VARU

VARU is a relatively recent project [6] aiming to be an integrated framework for VR, AR and Ubiquitous Computing. Thus, for a given application, there are three interaction *spaces* and it is possible to switch from a space (VR/ AR/ Ubiquitous) to another (object representation can be different) and it is possible for a user in a space to interact with a user in another space (since the whole is maintained to be consistent across the different spaces). Basically, objects are described by 3 levels of abstraction : Object *Class*, *Individual* and *Extension*. The *individual* is the instantiation of a *class*.

This individual has different representations depending on the interaction space. Each representation is called an extension of the individual. Each application has its object database with an *individuals* table and extensions table. When a user connects to the server, it is also considered as an individual and added to the object database with its extensions.

During the interaction, the different extensions of an individual are synchronized by the VARU server which hosts the object database and the object server. This way, other users are aware of other users actions and position (since a user is also an individual with its representations).

The main components of the VARU system are described in the Figure 4. As shown, every client has a *kernel* which links it to the VARU server. A client has also at least one space manager (VR, AR or UC) and can have other components for managing I/O devices, Display and streaming.

VARU uses CAIM middleware and UPnP protocol to connect to smart devices (such as smart door, smart light, smart camera...) and it uses VRPN for the interaction peripherals (e.g. joystick, tracker) management.

3. Frameworks Comparison

These criteria have been chosen on the current applications of VR. For example, security is not an important aspect yet because of the non-existence of wide used applications with public database access.

3.1 Ease of programming and reusability versus features

In the notion of ease of programming, we include the complexity of the application model, the ability for a developer to maintain

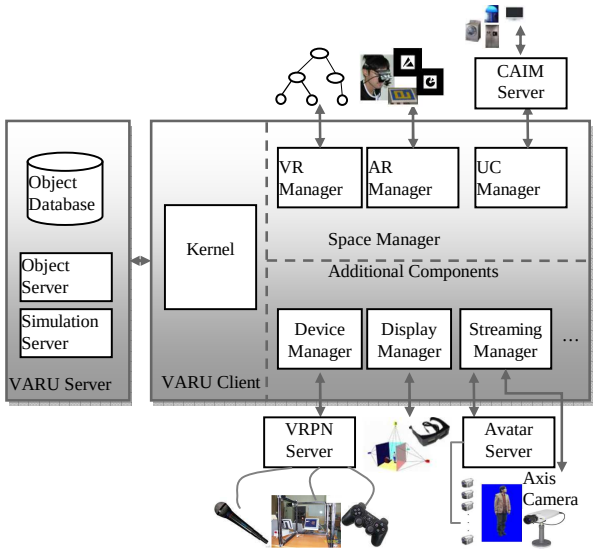

Figure 4. VARU main components

and reuse his code, and the ability to add a new functionality without having to change the overall system.

The main idea of all the described systems is to offer a framework for rapid prototyping of AR related applications. Thus, they are all Object-Oriented and most of them are based on a component approach (all except Tinmith evo-5 architecture). Reusability within the same framework is then a common strength for these systems.

From the ease of programming point of view, the simplest application model is the Studierstube's one being based on 3D concepts corresponding to the 2D Desktop metaphor. However, the simplicity of this model is balanced by the features offered that are not as numerous as other frameworks (such as DWARF and VARU).

An important point for the ease of programming is the existence of application authoring and monitoring tools. DWARF offers a dedicated tool to monitor and debug distributed services.

3.2 Scalability

Since we are covering distributed architectures, an important point is the ability to extend the number of the users (thus, network terminal nodes).

DWARF and MORGAN both use CORBA IIOP protocol which is widely used in systems of different sizes and has proven its high scalability potential even if some issues remain [12]. For example, load balancing is an important issue in scalability (in addition to caching and persistence to reduce network traffic). Until now, there has been not any load-balancing service in CORBA even if there are techniques to support the re-direction necessary for load balancing in a standard way [20]. An interesting aspect is that DWARF is entirely decentralized. The main proof of feasibility is that DWARF has been used in a gaming application (Herding Sheep [7]) with distributed tracking, calibration and user interface. However, when looking for new services, DWARF makes a broadcast that significantly increases network traffic (especially in a large scale application).

Studierstube distributed application objects are based on Distributed Open Inventor. The replication of the *Application Objects* restrains the level of scalability. Computational scalability can be achieved by introducing multiple application servers holding mutually exclusive sub scene graphs [5] but this increases significantly the number of *central nodes* and thus increases significantly the cost.

Concerning Tinmith, as said by Piekarski and Thomas in a Tinmith-evo5 architecture related paper [10], large scale distribution is not possible since it would require multicasting, which is not supported by the system at the moment. It is designed for a small number of users roaming in a wide area.

In the design of VARU, even if the architecture allows the addition of new users, the system requires a central server to synchronize different managers (AR manager, VR manager, UC manager) running on the multiple clients. The application of interior design [6] has been tested on a limited number of users and there is not any large scale distribution application using VARU.

3.3 Flexibility and interoperability

Flexibility refers to the ability of the framework to support different application scenarios running on different hardware (with different computational capabilities) and different software environment. We also include interoperability related features when referring to interoperable frameworks.

As said previously on the technical description of DWARF, the framework runs on multiple platforms and supports many programming languages. It can run on wearable computers, laptops and smart-phones as well and the architecture allows a wide set of applications. Dwarf can be integrated with studierstube [2] and has been used in several applications (ARCHIE, NAVI, CAR, SHEEP...) in different environments. Another important feature is the ability to reconfigure some parameters of the application at runtime.

MORGAN [8] runs on PC and smartphones and offers an API to reduce the programming complexity and to allow higher level programming. It addition, it offers a proxy for communicating with applications using other protocols. MORGAN also has its own rendering engine integrated with the framework. This last feature can also be a limitation for application developers who want to use other rendering engines.

Studierstube is quite portable on different hardware and OS especially with the new Studierstube ES platform dedicated to applications for hand-held devices [4]. The original Studierstube platform being based on OpenInventor, that is included as a rendering engine. However, Studierstube ES is renderer independent and runs on Windows CE and Symbian mobile devices with or without 3D graphics acceleration.

From the distribution point of view, Studierstube uses distributed OpenInventor (DIV protocol) for managing the distributed scene graphs. It is convenient to Studierstube applications but restricts the interoperability to OIV based applications (which is restricting but better than having a custom protocol). Studierstube ES management of networking is based on “Muddleware”, a communication platform for multi-user applications on lightweight terminals. It uses XML Document Object Model (DOM) which is a widely established data model for network computing. The choice of this model is, referring to its designers, due to the recursive definition of a tuple (which has child tuples as attributes of parent tuples), the readability of the model, and its match to many typical structures such as spatial hierarchical representations. XPath is used as a query/update language. The use of a server is mandatory to share data between two clients.

In the case of Tinmith, test Applications are mainly made for exterior environments. Like in DWARF applications, some parameters (like colors of gadgets, strings, positions...) of Tinmith platform applications can be reconfigured at runtime without restarting the application and without supporting an interpreted language.

From the distribution point of view, at the moment, Tinmith's custom protocol does not yet allow connections to applications using other protocols or to other AR frameworks.

Finally, the VARU framework takes advantage of the VRPN device server. The system supports a wide range of devices. The devices are configured to work with VRPN and the application simply connects to the server to get the data in a standardized way. VARU offers 3 *interaction spaces*. Thus, it can cover a lot of scenarios but these applications do not run well on mobile devices because of the heavy rendering engine [6].

From the distribution point of view, the clients and the VARU server both need to be configured to make the client's kernel access correctly to the server which adds a new user (and creates an individual and its extensions). This configuration being made by XML documents may be user unfriendly.

The study of these different reference frameworks summarizes the main characteristics, strengths and weakness of each framework taking in consideration differences in offered features. This study has allowed us to have a better vision on how we will improve our own framework (ARCS) especially from the distribution management point of view.

4. Improving ARCS

Before describing the improvements done to ARCS (which stands for Augmented Reality Component System) architecture, we need to briefly present the main concepts of the framework and how applications are built with it.

4.1 Previous Work

First, ARCS [3][18] is a component-based framework dedicated to AR. Its components, as classical components [19], can be configured and composed with other components. ARCS uses the signal/slot paradigm (borrowed from user interface libraries) to connect components to each other in order to make them communicate.

In ARCS, every application is described as a set of threads. Basically, each thread is controlled by a finite state machine which states represent a specific configuration of the application’s data flow. Such a configuration is called a *sheet* and contains configuration values for components as well as a list of signal/slot connections. Each change of state in the state machine results in a change of global configuration of components and hence reconfigures connection between components, that is to say the dataflow.

A simple example would be an application with one automaton (one thread), with a given number of *sheets* (eg. each *sheet* representing a given scenario). Here, the state machine’s role would be to switch from a scenario to another.

From the technical point of view, ARCS is written in C++ and is based on Qt Library which already implements the signal/slot

concept. ARCS supports XML as a scripting language to describe applications.

4.2 ARCS Middleware

4.2.1 Requirements

In its first version, ARCS was not distributed nor had network capabilities. Given the fact that our framework aims at supporting development of AR applications, adding network support and the network distribution functionality is mandatory to support state of the art applications.

As said previously, some frameworks use custom protocols while others use generic middleware to manage the application distribution. In particular, CORBA which is used in DWARF and MORGAN.

For our system (ARCS), we chose a specific custom architecture for two main reasons. The first one is the preservation of the signal/slot mechanism on which ARCS is based. The signal emitter is a client of the component having a slot receiving this signal. We also need to maintain the synchronous constraint inherent to this concept. The second reason is the ambivalent role of the components that can be, in our case, clients and services at the same time. This makes our architecture easier to setup, to maintain, to scale-up and more flexible since it can also work in a client/server manner. The two reasons cited above introduce the need of a specific and original architecture.

In addition to these two requirements of our own. We also identified three main requirements to answer the needs of developers of AR applications and help them to handle the specificity of such applications. These requirements inspired from Tokunaga & al. work [22] are: high-level abstraction, distribution and context-Awareness. They will be discussed further.

4.2.2 Architecture design

A first extension of ARCS architecture to allow distribution has been made. The chosen distributed architecture allows to transparently link remote components. The main idea is that an intermediary component is generated if a component is supposed to have some communication with another component on another machine. These intermediary components, built on the proxy design pattern [21], should include, among other data, all the connection data needed in order to communicate (remote host address, TCP port).

In the description of a *sheet*, if a component A needs to connect to a component B over the network, the component A is in fact connected (in signal/slot meaning) to the intermediary component on the machine where it is located (see figure 5 and 6). A *proxy slot* is created for each connection of a signal of this component with a slot of a distant component. A *proxy signal* is also created on the component B side to receive incoming remote signals and spread them locally.

Every remote component (*network component*) is considered as a service, it is attached to a specific component: the *network configurator* (the intermediate component) that instantiates *proxy slots* and *proxy signals*. A distant component can be client or server (it can be both). The *network configurator* receives on the *proxy signals component* all the signals destined to its local components, de-serializes them and forwards the signals to the actual destination components.

Here, we talk about de-serialization because once the connections between components and *network configurators* are created; we need to send the data over the network. Therefore, marshalling/unmarshalling mechanisms are setup within the *proxy slots* and *proxy signals*.

A *connection manager* (central component) lists all connections between components. It is used to set up the new data flow and activate different connections depending on the active *sheet*. Since all machines can be clients or servers, we decided to call the machine hosting the *connection manager* a *Master*. Other terminals are called *slaves*.

During the data transfer, once all connections have been set up, components on *slave* machines communicate without going through the master as shown in Figure 5 (the connections manager is only a repository of existing connections). The components can communicate in both directions and regardless of the location of the distant component (transparent communication).

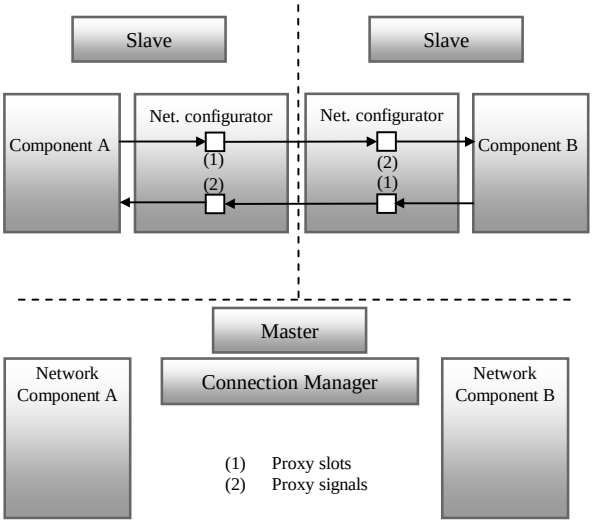

Figure 5. Remote connection between components in ARCS (Slave-Slave)

Figure 6 shows a Master – Slave communication. The component A is a local component for the master and it needs to send and receive signals/slots to/from component B on the slave machine. This need is specified in the XML application description. When running the application, the network component corresponding to component B is created on the master side and the network configurator to which it is attached is created on the slave side. Then, for each signal/slot connection, a proxy slot and a proxy signal are instantiated in the network component from one side and in the network configurator from the other side.

As implemented (and as described in this paper), this architecture answers our two first requirements. It keeps the component and signal/slot paradigm on which ARCS is based and offers the possibility to have components than can be clients and/or services.

It also meets the three requirements quoted from [22] and we will explain how it is done for each point:

High-level abstraction: AR applications often use different types of nodes (computers, mobile phones…) that can be specialized and running different operating systems. In our case, the goal was to offer a design generic enough to hide this complexity to the developer. In our solution, distributed components are localized

by a path system which allows more readable descriptions and makes application writing comfortable for developers. Also, the *network components family* in particular and the whole architecture in general is based on Qt library which is cross-platform. Finally, components being organized in a more abstract concept (*component family*), it allows developers even to enrich the framework itself with new components for a given family.

Distribution: as described above, the architecture is distributed and different application modules can be located on different network nodes. Distribution can be made on a very small granularity (at the level of a component). Thus, a low calculation capabilities mobile device can use a module running on a more capable computer to process data for example.

Context-Awareness: As said in ARCS description, the different scenarios of an ARCS application are described in its *sheets*. The required data about different software modules that interact with the application and the components handling hardware (e.g. camera capture component) are described within the sheets and are standardized within the components implementation (e.g. send camera images at the framerate of capture, if detection is not possible, send at a default framerate).

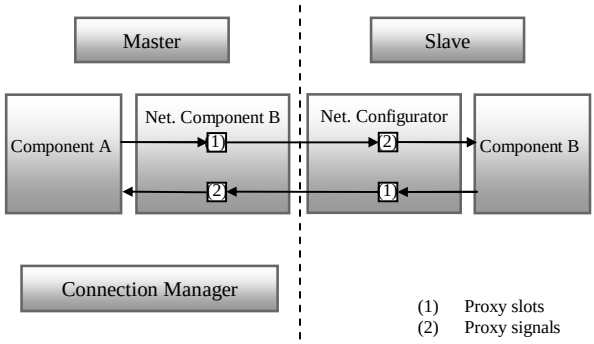

Figure 6. Remote connection between components in ARCS (Master-Slave)

4.2.3 Current Status and future work

This architecture has been tested and it works (even if detailed performance evaluation still needs to be done). Numerous toy tests have been made and they were conclusive. To allow the developer to have a global vision on his application, we also developed an architecture viewer (figure 7) reading application descriptions of the different nodes and building the final architecture at the machine and component level (to know which signal of which component is sending data to which slot of which component on which machine).

In addition, we also developed a tool (figure 8) for monitoring and debugging applications. It allows choosing ARCS application to run and displays the parameters of the application. A set of given values to the parameters is called a *profile*. These parameters can be changed at runtime.

Currently, in addition to qualitative and quantitative tests, an AR distributed application using ARCS and its middleware is being developed and is providing the most significant feedback for this architecture. The application consists of a SLAM (Simultaneous Localization And Mapping) application. Basically, its goal is to build 3D models of real environments explored simultaneously by multiple users which terminals collaborate to build a common 3D model.

ARCS middleware uses a custom protocol and as we noticed for studied frameworks, using a custom protocol is an important restraint for interoperability. We plan to use a more interoperable protocol in the future (ARCS design offers generic *components families* which makes it easy to make our middleware interoperable with other frameworks without having to redevelop it completely).

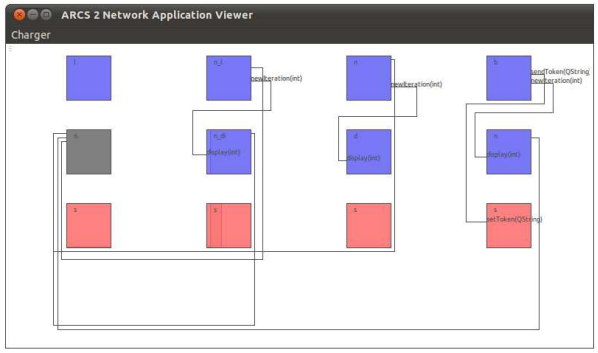

Figure 7. ARCS Network application viewer

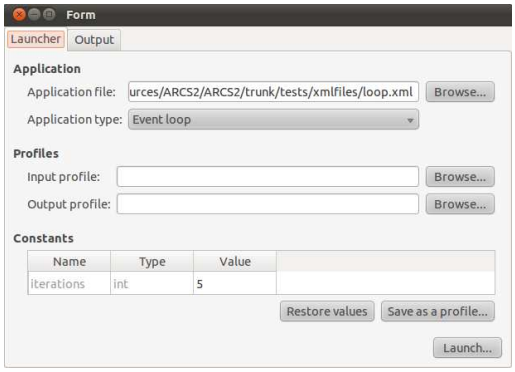

Figure 8. ARCS application wizard

We also noticed that the security is not the main concern when building current AR frameworks. Most applications run for a limited number of users in a relatively restricted area and it would be interesting to reinforce the security of the systems before considering large scale distributed AR applications in the future.

5. Conclusion

In this paper, we presented the design of a middleware dedicated to a framework for AR applications. We started by reviewing the most popular distributed AR frameworks. We described their application models and the way the distribution is managed within them. Even if these frameworks don't offer exactly the same features, we established general comparison criteria based on the needs of nowadays AR applications. From this comparison have emerged interesting ways to explore in order to advance in the ARCS platform extension especially on the network architecture and the protocol choice. The architecture has been designed and developed taking in consideration the existing framework, the complexity and specificity of this field, and the needs and comfort

of AR applications developers. In addition to the architecture itself, two ARCS application developer tools were presented. Finally, we also listed numerous possible improvements to our solution.

6. References

[1] M. Bauer, B. Bruegge, G. Klinker, A. MacWilliams, T. Reicher, S. Riss, C. Sandor, and M. Wagner. Design of a component-based augmented reality framework. In *Proceedings of the International Symposium on Augmented Reality (ISAR)*, Oct. 2001.

[2] M. Bauer, O. Hilliges, A. MacWilliams, C. Sandor, M. Wagner, J. Newman, G. Reitmayr, T. Fahmy, G. Klinker, T. Pintaric, and D. Schmalstieg. Integrating Studierstube and DWARF. In *International Workshop on Software Technology for Augmented Reality Systems (STARS)*, Oct. 2003.

[3] J. Didier, S. Otmane, and M. Mallem. A component model for augmented/mixed reality applications with reconfigurable data-flow. In *8th International Conference on Virtual Reality (VRIC 2006)*, pages 243–252, Laval (France), April 26-28 2006.

[4] S. Dieter and W. Daniel. Mobile phones as a platform for augmented reality. In *Proceedings of the IEEE VR 2008 Workshop on Software Engineering and Architectures for Realtime Interactive Systems*, Mar. 2008.

[5] A. Fuhrmann, G. Hesina, Z. Szalavari, L. M. Encarnacao, M. Gervautz, and W. Purgathofer. The studierstube augmented reality project. In *Presence: Teleoperators and Virtual Environments*, volume 11, Feb 2002.

[6] S. Irawati, S. Ahn, J. Kim, and H. Ko. Varu framework : Enabling rapid prototyping of VR, AR and ubiquitous applications. *IEEE VR*, Mar. 2008.

[7] A. MacWilliams, C. Sandor, M. Wagner, M. Bauer, G. Klinker, and B. Bruegge. Herding sheep : live system development for distributed augmented reality. *International Symposium on Mixed and Augmented Reality*, Oct. 2003.

[8] J. Ohlenburg, W. Broll, and A.-K. Braun. Morgan: A framework for realizing interactive real-time AR and VR applications. *IEEE VR*, Mar. 2008.

[9] J. Ohlenburg, I. Herbst, I. Lindt, T. Fröhlich, and W. Broll. The MORGAN framework: enabling dynamic multi-user AR and VR projects. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 166–169, New York, NY, USA, 2004. ACM.

[10] W. Piekarski and B. H. Thomas. Tinmith-evo5 - an architecture for supporting mobile augmented reality environments. *International Symposium on Augmented Reality*, Oct. 2001.

[11] W. Piekarski and B. H. Thomas. An object-oriented software architecture for 3d mixed reality applications. *International Symposium on Mixed and Augmented Reality*, Oct. 2003.

[12] Q. Gu, A. Marshall. Network management performance analysis and scalability tests : Snmp vs. corba. *IEEE/IFIP Network Operations and Management Symposium*, Apr. 2004.

[13] R. M. Weatherly, A. L. Wilson, B. S. Canova, E. H. Page, A. A. Zabek, M. C. Fischer. Advanced Distributed Simulation through the Aggregate Level Simulation Protocol. *Proceedings of the 29th Hawaii International Conference on Systems Sciences*, pages 407–415, 1996.

[14] IEEE 1278.1A-1998. Standard for Distributed Interactive Simulation - Application protocols, 1998.

[15] HLA IEEE 1516.1-2000 - Standard for Modeling and Simulation High Level Architecture – Federate Interface Specification, 2000.

[16] C. Endres, A. Butz, and A. MacWilliams. A survey of software infrastructures and frameworks for ubiquitous computing. *Mobile Information Systems Journal*, Amsterdam (Netherlands), Jav 2005.

[17] F. Zhou, H. B-L. Duh, M. Billinghurst - Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality* pp. 193--202, Washington, DC, USA, 2008

[18] J-Y. Didier, S. Otmane, M. Mallem - ARCS : Une Architecture Logicielle Reconfigurable pour la conception des Applications de Réalité Augmentée. *Technique et Science Informatiques (TSI), Réalité Virtuelle - Réalité Augmentée* 28 (6-7/2009):891-919, Jun-sep, 2009

[19] C. Szyperski, *Component Software - Beyond Object-Oriented Programming*, second edn, Addison-Wesley, Harlow, England, 2002.

[20] Object Management Group. http://www.omg.org

[21] E. Gamma, R. Helm, R. Johnson, J. M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, *Addison-Wesley Professional*, Nov, 1994.

[22] E. Tokunaga, A. van der Zee, M. Kurahashi, M. Nemoto, and T. Nakajima. Object-Oriented Middleware Infrastructure for Distributed Augmented Reality. In *Proceedings of the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (ISORC '03). May, 2003.