2014 IEEE International Conference on Services Computing

Multimodal interactions in dynamic, service-oriented pervasive environments

Philippe Lalanda, Laurence Nigay and Morgan Martinet
Laboratoire Informatique de Grenoble
Joseph Fourier University
Grenoble, France
firstname.name@imag.fr

***Abstract*—Heterogeneity and dynamicity of pervasive, service-based environments require the construction of flexible multimodal interfaces at run time. In this paper, we present how we use an autonomic approach to build and maintain adaptable input multimodal interfaces in smart building environments. We have developed an autonomic solution relying on interaction models specified by interaction designers and mediation components implemented by software developers. An interaction model is built around the notions of abstract device and abstract applications. The role of the autonomic manager is to build complete interaction techniques based on runtime conditions and in conformance with the predicted models.**

Keywords: multimodal interaction, service-oriented computing, mediation framework.

I. INTRODUCTION

Pervasive computing is slowly changing the way we interact with computers [1, 2] and is gaining more and more attention from industrial and academic sectors. This computing domain relies on the use of smart, communication-enabled devices integrated in our environment in order to provide users with added-value services. Research is particularly active in domains like smart homes or smart buildings where societal needs must be addressed. The purpose here is to assist in our daily activities in a natural and non-intrusive fashion. For instance, monitoring devices can be used to allow disabled or elder people to stay safely in their home, longer. Similarly, intelligent devices can be used to make our working environment more dedicated and efficient. For instance, rendering devices can help visitors to follow the right directions in an unknown building.

Pervasive devices are today becoming smaller and smarter. They fade away in the environment and appear as potential services rather than concrete hardware devices. They have the ability to communicate with each other, perform context-based functions, and manage themselves in order to stay operational. Such devices define ambient ecologies [3] with the ultimate goal to serve people. In this context, Weiser's exciting vision [1] where myriads of devices team up transparently to provide human beings with services of all sorts seem very reasonable! Indeed, plenty of devices have been designed and commercialized in the last few years. Surprisingly, however, applications seamlessly composing a number of devices are late to appear. We are in the strange situation where communication-enabled devices are not integrated and are still confined to limited usage.

Many devices and applications are today presented as software services [4]. The very purpose of the service-oriented approach is to build applications or interactions through the late composition of independent software elements, called services. Their capabilities are published at runtime and are subsequently discovered, chosen and called when needed. This is achieved within Service-Oriented Architectures (SOA) providing the supporting mechanisms for services description, publication, discovery, and invocation.

Service orientation brings interesting features that have made it popular in the pervasive domain. If carefully planned, using services can effectively support the development of environments with important (temporal and stochastic) variation. Weak coupling between consumers and providers reduces dependencies among composition units, letting each element evolve separately. Late binding and substitutability then improve adaptability: a service chosen or replaced at runtime is likely to better fulfil the consumer expectations. The fact is that a number of service-oriented platforms have appeared in the last few years and are more and more used to build pervasive applications in various domains [5].

However, interaction in dynamic, heterogeneous environments remains a challenging issue. The problem actually occurs in two ways: how could one use volatile, heterogeneous device to pilot an application and how could a given device be used to pilot volatile, heterogeneous applications. Precisely, it requires us to dynamically bind devices like mobile phones and service-based applications like media player for instance to allow an interaction. Composition is context aware in the sense that it relies on the available interaction devices and on the currently running applications. The situation can change anytime. It is not possible to anticipate all the possibilities in a design time composition.

In this paper, we present a flexible but controlled approach for the development and runtime management of input multimodal interfaces. Our approach relies on the autonomic management of interaction, realized with a mediation framework named Cilia. The paper is organized as follows. First we recall the key characteristics and requirements of pervasive multimodal interfaces. We then present our framework by describing the execution machine, the multimodal process and the autonomic manager. We conclude with an example that illustrates how the autonomic framework functions.

II. Multimodal Interaction

Pervasive environments lead people to reconsider the way they interact with electronic equipment. It seems, in particular, that graphical WIMP interfaces are too static and constrained to survive the proliferation of devices in our living environments. In saturated settings, users would better express their needs or desires with any available modalities, expecting the environment and its devices to react accordingly.

An interaction modality is defined as the coupling of an interaction device with an interaction language (i.e., a set of transformations of raw data from input devices). A multimodal interface is able to manage multiple interaction modalities. We focus on input multimodal interaction that allows the user to control a given application by using several interaction modalities. On the one hand, multimodality may involve the usage of multiple modalities in a coordinated manner in order to interact with a given application: for instance when combining gesture and speech commands in a complementary way. There are two possibilities for combining modalities, namely Redundancy and Complementary as defined by the CARE properties [6]. Combination of modalities requires fusion mechanisms [7]. On the other hand, multimodality can also refer to the existence of choice of modalities: the modalities are then equivalent for a given task and the user has the choice of the modalities for performing the task: this corresponds to the Equivalence property of CARE.

Multimodal interfaces have been shown to offer better flexibility and reliability than graphical WIMP interfaces [8]. Such interfaces fit well in the pervasive landscape. They offer a more natural way to interact with device-stuffed environments by means of speech, gestures or other modalities. Multimodal interfaces also allow users to employ a variety of devices to interact with an application, depending on the context (e.g. devices availability, reliability, user's mood, etc.).

The development of multimodal interfaces requires the integration of heterogeneous information sources, from devices to applications, in a timely fashion as defined by the Arch [9] interaction pattern. This involves a number of operations, including communication, synchronization, fusion, syntactic and semantic alignments. As schematized in Figure 2, these operations, often called mediation operations [10], demand some middleware support to be correctly developed, executed, and maintained.

As illustrated by Figure 1, specific component-based frameworks have been recently proposed to support the development of mediation operations for multimodal interfaces [11,12]. These approaches bring appropriate separation of concerns, clearly distinguishing functional aspects like data fusion and non-functional aspects like communication or synchronization. These proposals, however, are made for well-delimited environments where applications to be controlled and devices to be used are known in advance. They cannot handle highly dynamic environments where devices, applications, and the way multimodal interactions unfold, are rapidly evolving. More dynamic features are needed both at the design language level and the runtime execution framework level.

In a previous work, we developed component-based platform in order to better deal with environment unpredictability [13]. Here, we investigated the opportunistic composition of devices for a given application without having the complete interaction model that defines the multimodal interaction: This approach showed good results in emergency situations (e.g. a device breakdown) but clearly the resulting multimodal interaction may not be understood by the users. We need a more controlled approach and to put the user in the loop someway.

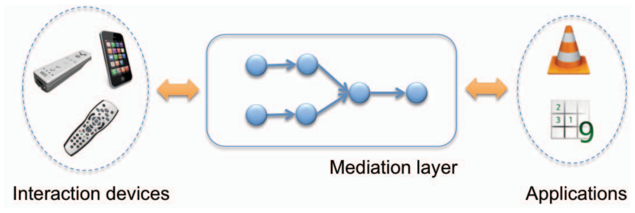

Figure 1. Mediation layer between devices and applications.

In the service computing field, Enterprise Service Buses (ESBs) have been defined in order to allow richer and better controlled interactions between clients and servers [14]. An ESB appears as a communication bus providing a unique interface to service providers and consumers. It can host mediation operations organized as processing chains transporting requests from consumers to providers and way back. Mediation chains are generally decomposed into specific components that implement mediation operations, which is an approach in conformance with the interaction framework mentioned before.

A number of products have been recently developed, including open source versions like Apache ServiceMix[1] or Mule[2] for instance. Many existing solutions are built on top of dynamic platforms like OSGi, which allows for runtime adaptation. Such platforms are resilient to changes in the environment and, obviously well adapted to service orientation. Current ESBs, however, are not really adapted to the management of multimodal interfaces. There are at least two reasons for that. First, current solutions are big in size. They target Information Systems, not pervasive infrastructures. Also, current solutions are still very technical and technology-driven. The development, deployment and management of mediation chains generally require highly skilled people. Last, but not least, current solutions are not autonomic. Adaptations cannot be decided and performed by ESBs themselves. So, a skilled administrator is needed.

To conclude, we believe that, one the great challenges of multimodal interfaces in pervasive environments, is then to build reliable and autonomic processing systems able to analyze and understand multiple communication means and reconfigure themselves in real-time. Solutions should be also implementable with the reasonable computed resources as generally available in pervasive settings.

[1] http://servicemix.apache.org
[2] http://www.mulesoft.com

III. Proposal

We have developed an autonomic approach to deal with multimodal interactions in service-oriented, pervasive settings. Autonomic computing characterizes the notion of a computer system that is able to adapt to internal and external change with minimal conscious intervention from the human [15, 16]. Here, the purpose is to create and manage multimodal interaction code depending on the running applications, the available devices, and the users' interaction preferences. Most devices and applications are service-based and are then highly dynamic.

Our proposition is illustrated by figure 2. It relies on the following elements:

- An introspectable and adaptable mediation framework, called Cilia, supporting the execution of multimodal processing. Cilia provides the necessary programming interfaces to create and update code at runtime and to sense internal and external context,
- Interaction models and policies defining the interaction opportunities in an abstract way and the user's preferences,
- Mediation components, stored in a code repository, that can be instantiated at runtime whenever needed,
- An autonomic manager, whose purpose is to actually create and manage the multimodal interaction code through the Cilia interfaces in a context-aware fashion, and following the interaction models and policies.

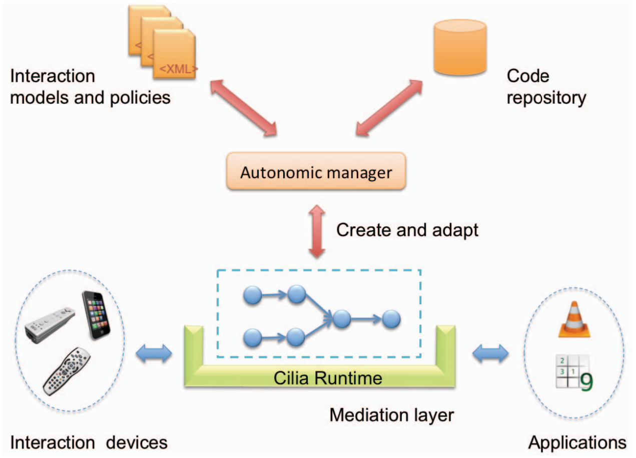

Figure 2. Proposal overview

Let us examine these different elements. In the first place, Cilia is a fully implemented mediation framework available in open source[3]. Its purpose is to simplify the work of developers and administrators by offering a well-defined set of abstractions to support the design, deployment and execution of mediation solutions. This framework hides many technical details and relies on autonomic capabilities to facilitate its administration. Technically speaking, Cilia is a domain-specific component model based on Java.

As introduced before, two kinds of interactions artefacts are prepared off-line:

- Interaction models define multimodal processing in an abstract way. They are made of abstract devices, mediation components and abstract applications. An abstract device regroups coherent interaction modalities. An abstract application represents a class of applications sharing similar services.
- Interaction policies define users' high-level preferences like the favourite devices, the preferred modalities, or the applications to be prioritized.

Mediation components implement mediation operations, in conformance with the Cilia development model. They are written in Java in a dedicated environment where generic Java classes can be reused or extended. Resulting components are stored in a code repository. One example of such a component is the Complementary component in order to combine two modalities.

The autonomic manager is the central element of our architecture. Its purpose is to manage the interaction code, from generation to destruction, in a context-aware, opportunistic fashion. By opportunistic, we mean that the autonomic manager puts in place the possible interaction modalities depending on the available devices and on the applications of interest.

The use of an autonomic manager limits the need for human intervention. Indeed, the autonomic manager uses the artifacts prepared off-line and on runtime information provided by the Cilia framework in order to meet its requirements. Precisely, it continuously scans the environment in order to get the available applications and interaction devices. It then manages the lifecycle of the mediation applications:

- It opportunistically creates code allowing interaction from available devices to applications,
- It dynamically updates running interaction code when execution conditions or users' preferences change,
- It deletes useless interaction code when devices or applications are no longer available.

Our approach is then based on a two-phase process. The purpose of the first phase is to prepare off-line a number of domain-specific artefacts that will be used at runtime to implement concrete interaction modalities. Interaction models and users' preferences are defined by experts in interaction whereas mediation code is implemented by software engineers, according to the specifications expressed in the interaction models.

The second phase addresses the runtime aspects. An autonomic manager handles the interaction code on top of the Cilia framework depending on the current environmental situation and in accordance with the defined interaction models and policies. The autonomic manager is influenced by high-level goals expressed by users.

[3] Cilia is available at https://github.com/AdeleResearchGroup/Cilia

IV. The Cilia framework

A. Main concepts

The Cilia mediation framework provides programming interfaces to dynamically create and assemble mediation components. It also provides facilities to update these assemblies at runtime. The programming interfaces handle domain-specific concepts. A number of technical aspects like synchronization are defined at a high level of abstraction and leads to low-level code generation.

Cilia is based on the notions of mediators and bindings. A mediator is a component implementing mediation operations like an aggregation, a transformation, a security function, etc. It declares a set of input and output ports. Input ports receive information to be treated and output ports provide the results of the mediation operation. Ports are the means to connect mediators and, thus, form mediation chains. Bindings are implemented as method calls by default. However, mediators can also be executed in different execution machines and bindings are realized in JMS (Java Message Service).

A mediator is made of three elements: a scheduler, a processor and a dispatcher. These are Java classes that can be developed independently. The scheduler deals with synchronization issues among other things. Its purpose is to store the received data in a buffer and to apply a triggering condition. When the condition is met, the buffer content is sent to the processor, which realizes the mediation operation. The result of this operation is sent to the dispatcher, which places the results in the output ports. Adapters are special mediators. They constitute the entry and exit points of a chain. They communicate with the resources to integrate.

More information about Cilia main concepts can be found in [17].

B. Cilia Runtime

The Cilia runtime environment is built on top of OSGi[4] and iPOJO [18]. OSGi provides the base mechanisms for modularity and dynamicity. iPOJO is the Apache service-oriented component model. It facilitates the development of dynamic component-based applications on top of OSGI through, in particular, the autonomic management of service dependencies. The platform also integrates a specific module, called RoSe[5] [19], which purpose is to constantly reflect the state of the computing environment in the execution machine. The RoSe framework captures services in the computing environment and reifies them as iPOJO components (proxies) in a local registry. It then manages the proxies' lifecycle, creating or deleting them as a reaction to arrival and departure of matching services. RoSe also provides advanced facilities to specify the services of interest according to different properties like type, protocol, or number. ROSE currently handles several service-oriented protocols including Web Services, DPWS, UPnP, Zigbee and Bluetooth.

[4] www.osgi.org

[5] RoSe is available at https://github.com/AdeleResearchGroup/ROSE

As illustrated by figure 3, the Cilia runtime provides monitoring and adaptation interfaces or touchpoints. These touchpoints require deep knowledge of the managed elements. For that reason, the Cilia framework also allows the automatic construction of a configurable knowledge base storing runtime information about the mediation chains and the execution context. This knowledge base provides a model of runtime phenomena, with trends and past data, and is intended for use by autonomic managers. This model is causal in the sense that modifications made on the model representation are reflected on the Cilia runtime and vice versa. Using this knowledge module is a very convenient way for domain engineers to create autonomic managers. Management is done through high level programming interfaces and does not demand to be familiar with the intricacies of Cilia.

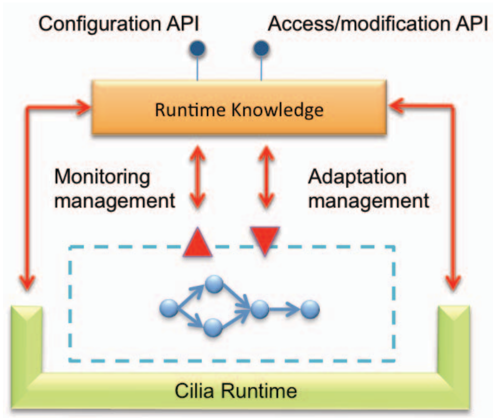

Figure 3. Runtime information provided by Cilia

Monitoring is flexible and configurable. It can be controlled in a dynamic way. This means that Cilia monitoring can be activated or deactivated globally. It also means that the elements to be monitored, and the way they are monitored, can be configured without interruption of service. Monitoring provides information about the mediation chains under execution, about the execution machine and about the external context (through RoSe). The dynamics of the mediation chains is encoded as state variables. Their values, called measures, are kept in circular lists in order to keep records of the past. The size of the circular lists is configurable and can be changed at runtime. From the knowledge thus acquired, adaptations can be decided. Specific mechanisms have been designed in order to protect ongoing computations while some code is being updated. Precisely, a quiescence mechanism has been implemented to preserve control flows and save the messages being processed.

A Cilia application can thus be complemented with an autonomic manager. It is to be understood however that this manager is domain-specific. In our case, the purpose of the autonomic manager is to create and adapt the multimodal interactions, using the dynamic capabilities of the underlying component model (Cilia) and its knowledge about interaction management.

V. Detailed approach

The big challenge tackled in our work is to dynamically link devices and applications that are not known at design time and to do so in a way that is understood and accepted by users. Our approach relies on an autonomic manager using design-time artefacts and runtime models to create and update multimodal interaction code in service-oriented environments. In particular, interaction models are defined off-line. They are based on the notion of abstract devices, mediation components, and abstract applications. This approach is in accordance with the architectural pattern of the Arch model (see section II). Let us now describe these different elements in the coming sections.

A. *Abstract device*

An abstract device groups a set of devices according to how they can be used as part of a modality. These modalities can be used together in a combined way (combined modalities) or not (atomic modalities). For instance, an abstract device may comprise a pad and a gyroscope. An interaction modality may rely on the pad and the gyroscope or on the pad only. Abstract devices focus on high-level interaction modalities and ignore low-level technological aspects. In our approach, these abstract devices are defined in terms of the following:

- A name characterizing their purpose,
- A set of interaction modalities, which are described in a unifying language,
- A set of output ports providing data,
- A set of properties, identified by their names and types, used to specify static and dynamic information about the devices.

Abstract devices are dynamically associated with one or several concrete interaction devices at runtime. These concrete devices are manipulated by users to interact with their pervasive environment. A concrete device may correspond, for instance, to a Wii mote or to a smart phone. Properties attached to abstract devices are used to widen or to limit the runtime selection of concrete devices. Properties can actually be very specific about the expected concrete devices. Nothing prevents the interaction designer from using a concrete device (e.g., an iPhone 5) when specifying an interaction model. The idea, however, is not to provide extensive information and to leave sufficient room for runtime selection.

Once a concrete device is selected at runtime, some code has to be generated or instantiated in order to effectively communicate with that device and link it to the appropriate mediation chain. As will be seen in the next section, such dynamic code generation relies on the RoSe framework and on the notion of communication proxy.

B. *Abstract application*

An abstract application represents a class of applications sharing similar functions or tasks. Formally, an abstract application is a service description that is independent of any given implementation technology. It retains the major features of the service orientation and ignores low-level technological aspects. For instance, an abstract application can be a media player described with the basic set of functions/tasks provided by such software.

In our model, an abstract application is defined in terms of the following:

- A set of functional interfaces specifying the provided functionalities. An interface can define either a set of operations to be invoked or a set of ports to be used for data flows.
- A set of properties, identified by their names and types, used to store static and dynamic information about the applications. Such information can be related to functional or non-functional aspects.

Abstract applications are dynamically associated with concrete applications at runtime. Concrete applications are the *actual* applications, dynamically available in the pervasive environments, and used by users to get services. A concrete application can be, for instance, the VLC media player. As for the devices, properties are used to select the appropriate applications at runtime. The variability expressed by the properties makes room for adaptation to a greater or lesser extent. However, as for abstract devices, the guiding principle is still to bring flexibility at runtime while preserving some invariants. So, it is expected not to be restrictive when setting properties.

In our approach, concrete applications are implemented using service-based technologies. For instance, a concrete application can be exposed as a Web Service, an UPnP service or a DPWS service. Here again, when an application is selected at runtime, some code has to be instantiated. APIs between abstract and concrete applications can be different and alignments are often necessary. Proxies, managed by RoSe, are inserted in order to handle these differences and properly invoke the programming interfaces of the selected concrete application.

C. *Interaction models*

Interaction models express how to process data from abstract devices and route them to programming interfaces pertaining to abstract applications. As illustrated by Figure 4, abstract devices and abstract applications are respectively the entry points and exit points of Cilia mediation chains. The inside of the chains is made of mediation components.

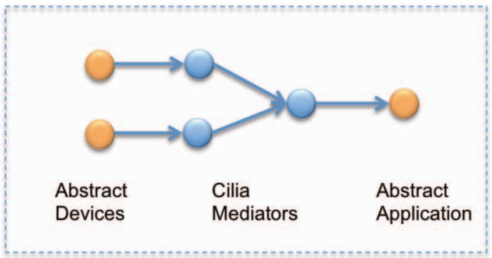

Figure 4. Interaction model

A mediation component realizes one or several mediation operations like data transformation or data fusion. In our approach, mediation components are implemented as Cilia components and specified accordingly. Contrarily to abstract devices and applications, they are concrete software artefacts, entirely developed in Java by software engineers.

Interaction models can be more or less sophisticated (combined or atomic modalities), with various numbers of abstract devices and applications, different possible paths, and so on. In our approach, it is up to the interaction designers to define the appropriate level of variability of the interaction models depending on their intent, the users' preferences and the problem at hand.

D. *Autonomic manager*

The role of the autonomic manager is to dynamically create and maintain interaction code while meeting the users' expectations. To do so, the autonomic manager relies on interaction models and policies providing the knowledge necessary to set up the appropriate processing code given the runtime situation.

The autonomic manager implements a control loop, typically divided into the following tasks [20]:

- A monitoring task follows changes in the execution context. It reports the devices and applications availability and, also, the performance of the running mediation chains and underlying execution machine,
- An analysis task compares the interaction modalities actually implemented and those that could be implemented given the execution context. It decides whether modifications are necessary or not,
- A planning task that is rather simple in our case. It schedules the management actions decided by the analysis task,
- An execution task invokes the Cilia APIs and checks that the requested actions are actually implemented as expected.

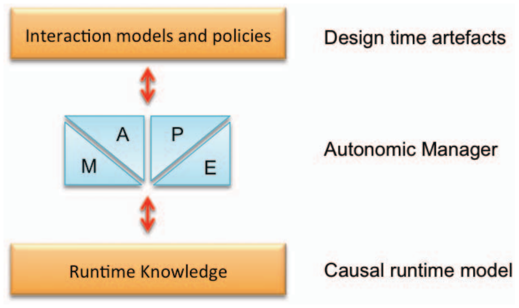

Figure 5. Model-based autonomic manager.

As illustrated by Figure 5, our approach is based on the complementary notions of model and software architecture. Architectural models are used to define the goals of the autonomic manager in abstract terms and also to present a simplified, focused view of the runtime situation. It is to be noted here that the monitoring and execution tasks are realized via the runtime model, i.e. the Cilia knowledge base. The job of the autonomic manager here is *simply* to dynamically configure the monitoring policies and to trigger adaptation on the Cilia knowledge base, which is causal (see section IV).

The advantage of this approach is that the code of the autonomic manager remains focused on the domain at hand, the management of multimodal interaction, and is not burdened by low level technical considerations.

The management of volatile service-based devices and applications is performed by the RoSe framework, which is part of the Cilia execution machine. Precisely, RoSe tracks the appearance and departure of services and, in response, creates or deletes proxies. RoSe actually relies on the service-oriented architectural pattern and clearly decouples discovery and communication. This allows the dynamic extension of the framework: protocols and proxy management policies can be updated anytime. Also, RoSe only tracks devices meeting some filtering LDAP conditions regarding for instance protocols, addresses, provided services, properties, etc. This is extremely important in environments saturated with smart, communicating elements. Here again, filtering conditions can be updated anytime without service interruption.

Proxies take the form of service-oriented iPOJO components. They are thus made available as services in the execution machine. It is to be noted that proxies are not generated but downloaded from a dedicated code repository. Thus, a device can be used in an interaction only if a proxy has been developed off-line and made available in the repository.

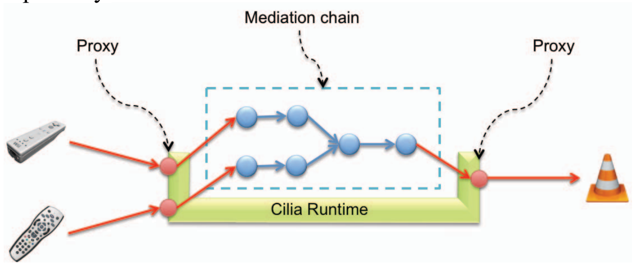

Figure 6. Executable chain.

An important task of the autonomic manager is to link communication proxies to the mediation chains. This is done in two steps. First, the autonomic manager configures the RoSe framework so that it looks for the appropriate services and creates corresponding communication proxies. Second, when expected proxies are created, which the autonomic manager can then link to the abstract devices and abstract applications, as shown in Figure 6. At that point, a, interaction processing chain is created and can be started.

VI. VALIDATION

A. *Performances*

We first evaluated the performance overhead of our framework regarding monitoring and adaptation. The diagram of Figure 7 presents the time needed for data to traverse a mediation chain, with and without monitoring. To obtain these figures, we sent up to 200 messages in the mediation chains through scripts [21]. It clearly appears on the figure that monitoring overhead remains stable when the number of messages increases. A difference in performance of a few milliseconds has no impact at all in our context.

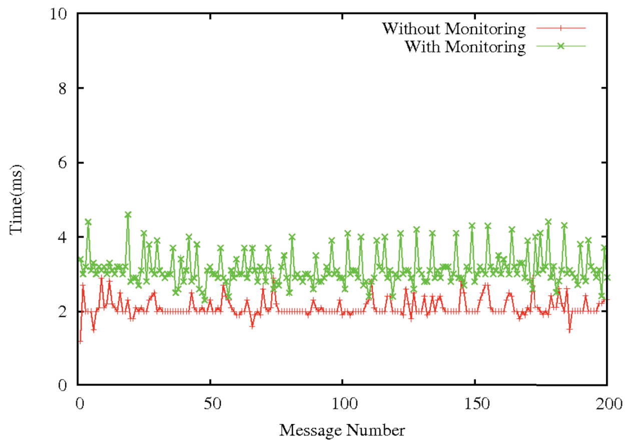

Figure 7. Cilia monitoring cost.

We also measured the processing times needed to update different aspects of a mediation chain. To do so, we have developed a specific autonomic manager whose purpose was to perform some pre-defined updates when receiving a signal. Table 1 shows the resulting findings.

Operation	Time (ms.)
Mediator creation	130
Binding creation	60
Binding removal	50
Mediator removal	115
Mediator replacement	200
Mediator reconfiguration	7

Table 1. Cilia dynamic adaptation overhead.

The processing time engendered by updates varies a lot depending on the nature of the modification. Reconfigurations and binding manipulations are rather cheap. Contrastingly, creations and replacements of mediators are much more costly, which seems understandable, given the cost of the quiescence mechanism. These performance overheads are tolerable in our context.

B. Scenario

For testing the proposed approach, we have developed different scenarios illustrating the appearance of applications and interaction devices. In terms of multimodal interaction, the example illustrates a case of equivalence of modalities as defined by the CARE properties for multimodal interaction. While the example considers the equivalence of two modalities (one modality based on a TV remote controller and one based a controller for a video game console), our framework also supports the complementarity of modalities by providing fusion mechanisms of two modalities, for example a pointer for defining a point on a map and two buttons for defining the zoom factor in the case of a navigation task.

We considered two applications: VLC[6] and KSudoku[7]. We did not develop the two applications but reused them. It shows that we can integrate existing applications in our platform and control them with different interaction devices. The two applications can be accessed through an inter-process communication, called D-Bus [8] protocol that is supported by our platform. A rather simple proxy is created by a developer for each application, without adding any code to the two existing applications. For interaction, we considered two devices: a TV remote controller, also called BDRC, and a video game console controller (Wii Remote, also called Wiimote in short).

For guiding the autonomic manager, we have defined interaction models for the two applications and the two types of devices. Remember, the interaction models are expressed in terms of abstract devices and applications.

Then, we played different scenarios. Let us examine one of them: the VLC media player is started in an environment simply composed of a TV remote controller. Therefore an interaction is generated: VLC can be controlled by the BDRC. When KSudoku is started, the environment is composed by one device and two applications. A policy leads the autonomic manager to stop the interaction with the first application, and bind the devices to the last discovered application. Finally, when the Wiimote is discovered, the current interaction is enhanced in order to enable the Wiimote to also control KSudoku, while interaction using the DBRC is still possible: two modalities are then equivalent for controlling KSudoku

Figure 8 presents one example of a mediation chain between BDRC and VLC created at runtime. When the user will press the button "zero" of the TV remote control BDRC, the volume will be set to mute. By pressing the button "Pause" of BDRC, the movie currently showed will be stopped.

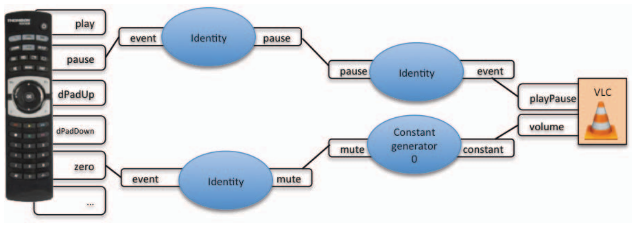

Figure 8. Generated mediation chain for VLC and BDRC interaction.

Figure 9 presents another example of a mediation chain generated at runtime between the Wiimote and VLC. The "x" port of the Wiimote is bound to the volume port of VLC, and the button "a" of the Wiimote, to the port "playPause" of VLC. By moving the Wiimote horizontally, the volume will be increased or decreased, and the user must select the button "a" of the Wiimote to stop the movie.

[6] www.videolan.org/vlc

[7] games.kde.org/game.php?game=ksudoku

[8] http://dbus.freedesktop.org

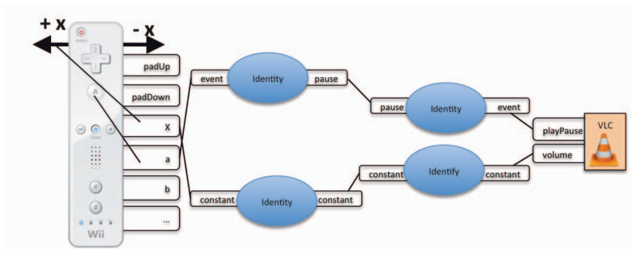

Figure 9. Generated mediation chain for VLC and Wiimote interaction.

VII. Conclusion

In this paper, we have presented an approach to the construction of flexible multimodal interfaces at run time in pervasive, service-oriented settings. We believe that the heterogeneity and dynamicity of such environments call for autonomic solutions with solid software engineering foundations. Precisely, we have developed a modular approach clearly separating service management, mediation operations and the overall administration of the infrastructure.

The framework being fully operational, as further work, we will first perform experimental evaluations with users. Such experiments will enable us to enrich the autonomic manager by identifying new policies. For example, the adaptation is currently realized without learning from the users' inputs. However, several cases would obviously leverage the learning. For example, if a button is never used by a user, the framework could propose to bind this button to another function. The usage of an autonomic architecture will ease the machine learning process because sensing and effecting are already done. Moreover based on our framework we will study an important aspect of dynamic multimodal interfaces that is how to make observable by the users the performed changes in multimodal interaction for example due to a new discovered device.

Our general research direction is to make the autonomic manager observable and controllable by the users by defining different levels for tuning the autonomic capacity of the framework and therefore making the user in control of her/his pervasive environments.

VIII. References

[1] Weiser, M. 1991. The computer for the 21st century. *Scientific American*, 265(3), 66-75

[2] Satyanarayanan,M. 2001. Pervasive computing: vision and challenges. *IEEE Personal Communications*, Vol. 8 (August 2001), 10-17

[3] C. Escoffier, J. Bourcier, P. Lalanda, J. Yu, "Towards a home application server", 5th IEEE Consumer Communications and Networking Conference, CNNC 2008. Pages 321-325, 2008.

[4] Papazoglou,M. P. and Georgakopoulos,D. 2003. Service-Oriented Computing: Introduction. *Communications of the ACM* 46, 10 (October 2003), 24-28.

[5] Escoffier C., Chollet S. and Lalanda P., "Lessons learned in building pervasive platforms", The 11th IEEE Consumer Communications and Networking Conference, Las Vegas, January 2014.

[6] Coutaz, J., Nigay, L. Salber, D., Blandford, A, May, J., and Young, R.M. 1995. Four easy pieces for assessing the usability of multimodal interaction: the CARE properties. In Proceedings of IFIP TC13 Interantional Conference on Human-Computer Interaction INTERACT, (June 1995), Chapman & Hall, 115-120.

[7] Serrano, M, and Nigay, L. 2009. Temporal Aspects of CARE-based Multimodal Fusion: From a Fusion Mechanism to Composition Components and WoZ Components. Proceedings of International Conference on Multimodal Interfaces ICMI, (Nov. 2009), ACM, 177-184.

[8] Oviatt, S. 2007. Multimodal interfaces. *Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications*. L. Erlbaum Assoc. Inc., Hillsdale, NJ, USA. (2007), Chap. 14, 286-304.

[9] The UIMS tool developers workshop. 1992. A metamodel for the runtime architecture of an interactive system. *SIGCHI Bulletin* 24, 1 (January 1992), 32-37

[10] Wiederhold, G. and Genesereth, M. 1997. The Conceptual Basis for Mediation Services. *IEEE Expert: Intelligent Systems and Their Applications* 12, 5 (September 1997), 38-47.

[11] Bouchet, J., Nigay, L. and Ganille, T. 2004. ICARE software components for rapidly developing multimodal interfaces. Proceedings of International Conference on Multimodal Interfaces ICMI, (October 2004), ACM, 251-528.

[12] Serrano, M., Nigay, L., Lawson, J.-Y. L., Ramsay, A., Murray-Smith, R. and Denef, S. 2008. The openinterface framework: a tool for multimodal interaction. Proceedings of SIGCHI Conference on Human Factors in Computing Systems Extended Abstracts CHI EA, (April 2008), 3501-3506.

[13] Avouac P.A., Lalanda P. and Nigay L., "Autonomic management of multimodal interaction: DynaMo in action", Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems, 2012.

[14] Hérault C., Thomas G. and , Lalanda P., "Mediation and Enterprise Service bus: a position paper", Proceedings of the First International Workshop on Mediation in Semantic Web Services (MEDIATE 2005), pp. 67-80, 2005.

[15] Horn P., "Autonomic Computing: IBM's Perspective on the State of Information Technology", IBM, 2001.

[16] Lalanda, P., McCann, J., and Diaconescu A., A. *Autonomic Computing: Principles, design and implementation*. Springer Verlag, London, 2013

[17] Garcia, I., Pedraza, G., Debbabi, B., Lalanda, P., Hamon, C. Towards a service mediation framework for dynamic applications. In *Proc. APSCC 2010*, IEEE Asia-Pacific Services Computing Conference, IEEE (2010), 3-10.

[18] Escoffier, C., Hall, R. S. and Lalanda, P. 2007. iPOJO: an Extensible Service-Oriented Component Framework. *Proc. of SCC 2007*. IEEE Computer Society, Washington, DC, USA, 474-481.

[19] Bardin J., Lalanda P., Escoffier C., Towards an automatic integration of heterogeneous services and devices, IEEE Asia-Pacific Services Computing Conference, pp. 171-178, 2010

[20] Kephart, J. O. and Chess, D. M. 2003. The vision of autonomic computing. Computer 36, 1 (Jan.), 41–50

[21] Lalanda P, Hamon C., Escoffier C., and Leveque T., "iCasa, a development and simulation environment for pervasive home applications", The 11th IEEE Consumer Communications and Networking Conference, Demonstration, Las Vegas, January 2014.