Autonomic Management of Multimodal Interaction: DynaMo in action

Pierre-Alain Avouac, Philippe Lalanda and Laurence Nigay
Université Joseph Fourier Grenoble 1
Laboratoire d'Informatique de Grenoble LIG UMR 5217, Grenoble, F-38041, France
{Pierre-Alain.Avouac, Philippe.Lalanda, Laurence.Nigay}@imag.fr

ABSTRACT
Multimodal interaction can play a dual key role in pervasive environments because it provides naturalness for interacting with distributed, dynamic and heterogeneous digitally controlled equipment and flexibility for letting the users select the interaction modalities depending on the context. The DynaMo (Dynamic multiModality) framework is dedicated to the development and the runtime management of multimodal interaction in pervasive environments. This paper focuses on the autonomic approach of DynaMo whose originality is based on partial interaction models. The autonomic manager combines and completes partial available models at runtime in order to build multimodal interaction adapted to the current execution conditions and in conformance with the predicted models. We illustrate the autonomic solution by considering several running examples and different partial interaction models.

Author Keywords
Multimodal interaction; Autonomic computing; Model-based engineering; Service-oriented components.

ACM Classification Keywords
D.2.2 Software Engineering: Design Tools and Techniques – User interfaces.

General Terms
Algorithms; Human Factors.

INTRODUCTION
Pervasive environments lead people to reconsider the way they interact with digitally controlled equipment. Indeed, facing the proliferation of communicating devices in the environments, the users will express their needs or desires with any available interaction modalities, expecting the environment and its equipment to react accordingly [26]. As motivated in [2], multimodal interaction fits very well in pervasive environments because multimodality offers (i) a natural way to interact with equipment including gesture, speech and direct manipulation [6] (ii) flexibility in letting the users select the modalities according to different contexts (tasks to be performed, interaction devices availability, social context, etc.).

In order to develop and autonomically manage multimodal interaction in service-based pervasive settings, we designed and developed the DynaMo framework. Autonomic in the context of DynaMo means that management decisions are taken and realized by the framework itself. The overall architecture of the underlying platform of DynaMo is described in [2] and a simple scenario illustrating the appearance of an interaction device and therefore a new interaction modality is presented in [3]. After a declarative description of the underlying platform and its software layers in [2], this paper focuses on the management of multimodal interaction by the autonomic manager (i.e., the platform in action). We describe a complete example highlighting the dynamic aspect of multimodal interaction

Based on the dynamic capabilities of the underlying platform, the DynaMo autonomic manager creates and modifies the multimodal processes at runtime. A multimodal process for input multimodality defines the interpretation function and is made of a sequence of input transformations: Information acquired by input digital channels (physical interaction devices) is transformed and abstracted to obtain a meaningful application task through multiple process activities characterized with four intertwined ingredients: level of abstraction, context, fusion/fission, and parallelism [21]. In order to create and update these multimodal processing chains, the autonomic manager contains domain-specific knowledge. The originality of our approach relies on the definition of partial interaction models in order to specify the autonomic manager knowledge and constraints. The interaction models are characterized as partial with respect to the complete multimodal transformation chain which ranges from raw data captured by devices to elementary tasks. A partial interaction model will thus define a sub-part of this transformation chain. In this paper we present these partial interaction models organized according to the ARCH software architectural model [1] and illustrate how they are used by the autonomic manager.

The structure of the paper is as follows: first, we motivate the adopted approach for designing DynaMo at the intersection of two domains, multimodal engineering and pervasive computing. We then recall the overall architecture of DynaMo fully specified in [2] before describing and illustrating the autonomic manager and the manipulated partial interaction models.

DYNAMO: PERVASIVE COMPUTING FOR MULTIMODAL ENGINEERING

On the one hand, several frameworks have been defined for developing multimodal interaction including ICON [11], ICARE [5], OpenInterface [24], Squidy [18] and MUDRA [16]. Such frameworks are mainly based on a component-based approach, which allows the easy and rapid development of multimodal interfaces. Indeed the designer specifies multimodal interaction dedicated to a given task of the interactive system under development by assembling components, the corresponding code being automatically generated. Such frameworks predominantly take on a data-flow approach that has been shown to be adapted for specifying multimodal interaction: indeed the assembling of components defines the data-flow from interaction devices to application tasks. Figure 1 presents an example of a data-flow as an assembly of generic and tailored software components from the OpenInterface framework [23].

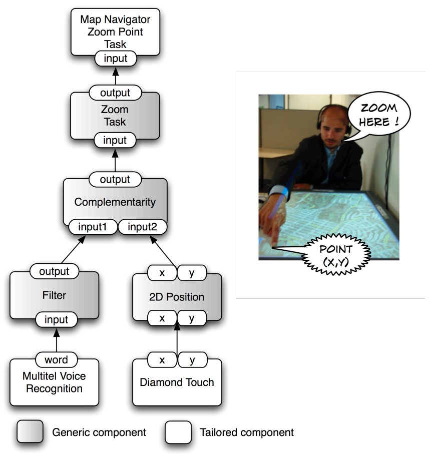


Figure 1: Multimodal processing as an assembly of components describing the data-flow from devices to tasks. (from [23]).

Some existing frameworks include a graphical editor that allows direct manipulation and assembling of components in order to specify multimodal interaction. Figure 2 presents a screenshot of the Squidy graphical editor [18].

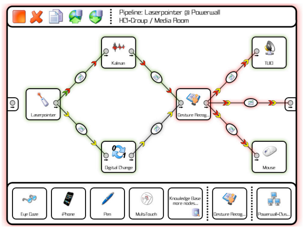
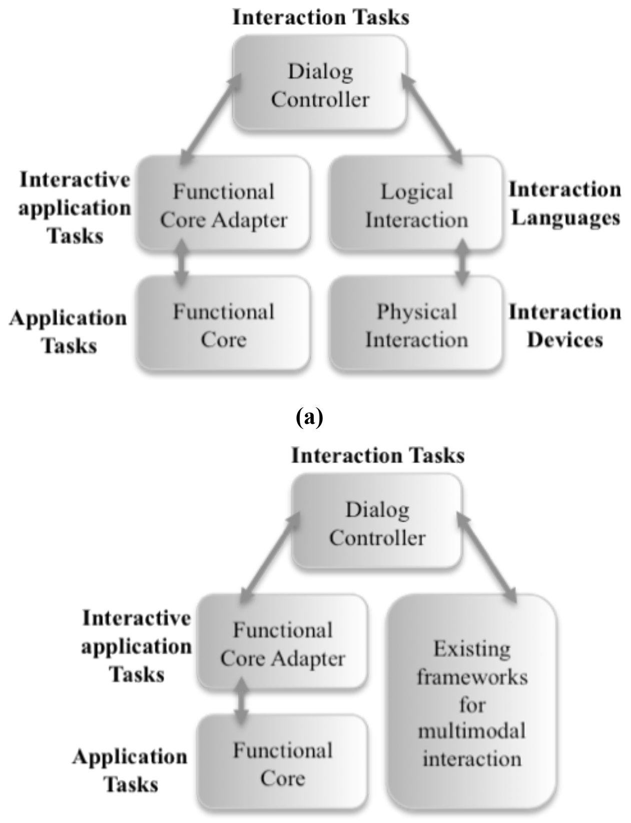


Figure 2: Screenshot of the graphical editor of Squidy (from [18]).

To fully understand the scope of these frameworks we show in Figure 3 where the corresponding code is located within the complete code of the interactive multimodal system structured along the ARCH software architectural model [1].



(b)

Figure 3: (a) Multimodality and the ARCH software architectural model: An interaction modality is defined by the couple: (device, language). The Physical Interaction component is device dependent and the Logical Interaction component is device independent but language dependent. (b) Existing frameworks for multimodal interaction within an ARCH software architecture.

Finally some frameworks support the dynamic discovery of input devices. Such frameworks therefore provide some flexibility by defining adaptable multimodal interaction. Such adaptation is made possible by defining at design time equivalence modalities for a given task. Equivalence of modalities for a given task is defined in [21] as one of the CARE properties. But multimodal adaptable interaction is completely defined at design time. One example is the COMET interactors [10]. COMET interactors are dedicated to plastic user interfaces. In particular for input multimodality, a COMET interactor includes a facet called *physical model* that describes input and output. For input, several equivalent devices can be defined at design time. At runtime the user can then switch between modalities.

Such approaches, however, are made for well-delimited environments where application tasks to be controlled and interaction devices to be used are known in advance. The existing frameworks cannot handle highly dynamic environments where devices, applications, and the way multimodal interactions unfold, are rapidly evolving. For multimodal interaction not fully defined at design time, more dynamic features are needed both at the design language level and the runtime execution framework level. Figure 4 schematizes the highly dynamic context that we addressed with DynaMo according to the ARCH software architectural model.

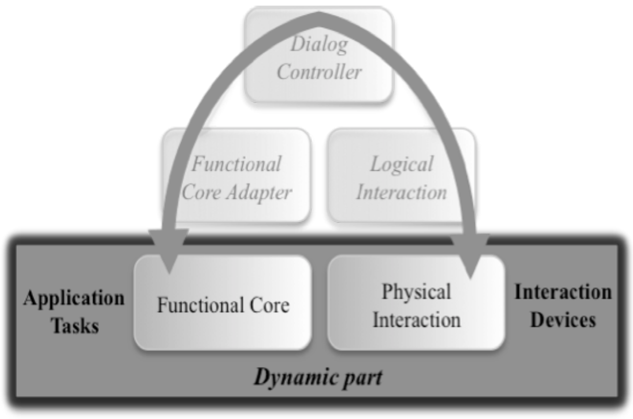


Figure 4: Dynamic context addressed by the DynaMo framework.

On the other hand, pervasive computing is influenced by advances in service-oriented computing [22] whose purpose is to build systems through the late composition of independent software elements called services. Service-oriented computing allows us to manage highly dynamic pervasive environments. Our approach then considers service-based applications (Functional Core of ARCH in Figure 4) and service-based interaction devices (Physical Interaction of ARCH in Figure 4). Multimodal interaction with pervasive applications is then a truly illuminating case. It requires us to dynamically bind service-based interaction devices like mobile phones, TV remote controls and wiimotes and service-based applications like media players and games. 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 at anytime. It is not possible to anticipate all the eventualities in a design time composition, even through abstraction. Multimodal interactions between service-based devices and applications require the integration of heterogeneous information sources in a timely fashion implying a number of operations, including communication, synchronization, fusion, syntactic and semantic alignments as defined in the previous section (i.e., the interpretation function [21]). In the pervasive computing field, these operations are called mediation operations [27], and demand some middleware support to be correctly developed, executed, and maintained. Enterprise Service Buses (ESBs) have been developed in order to allow richer and better controlled interactions between clients and servers. 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 back. Mediation chains are generally decomposed into specific components that implement mediation operations. A number of products have been recently developed, including open source versions such as Apache ServiceMix or Codehaus Mule for instance. Many existing solutions are built on dynamic platforms like OSGi, which allows for runtime adaptation.

Current ESBs are not 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. Towards this goal of flexibility to be managed by autonomic managers, we have designed and developed a service mediation framework for dynamic applications called Cilia [14]. We added mechanisms in order to build adaptable mediation solutions (i.e. adaptable multimodal interpretation processes) based on structural and behavioral reflection [13] that can be used by external managers in charge of performing adaptations at runtime. Cilia is the mediation framework of DynaMo and is built on top of a service integration platform called iPOJO [12].

The design and development of the DynaMo framework is therefore based on results from both multimodal engineering and pervasive computing. DynaMo combines recent advances in component-based multimodal engineering, service-oriented component engineering as well as adaptable service mediation mechanisms. In the following sections, we recall the overall architecture of DynaMo before describing the model-based autonomic manager and its models.

DYNAMO: OVERALL ARCHITECTURE

As shown in Figure 5, DynaMo is made of three main parts.

First DynaMo relies on a service integration platform (based on OSGi and IPOJO [12]) that monitors the environment in order to trace any computing evolution. In particular the ROSE module [4] captures services (i.e., interaction devices or applications) in the computing environment and reifies them as iPOJO components in an advanced service registry. Various protocols are supported including Web services, Zigbeee, Bluetooth, UPnP and DPWS.

Second DynaMo includes a lightweight component-based mediation framework called Cilia as introduced above. Cilia allows the execution of adaptable multimodal processes. An assembly of Cilia mediators also called Cilia mediation chain corresponds to the data-flow from interaction devices to application tasks as specified at design time with the existing multimodal frameworks (Figures 1 and 2).

The third constituent of DynaMo is a model-based autonomic manager that creates and adapts the multimodal interaction using the dynamic capabilities of the underlying Cilia component model.

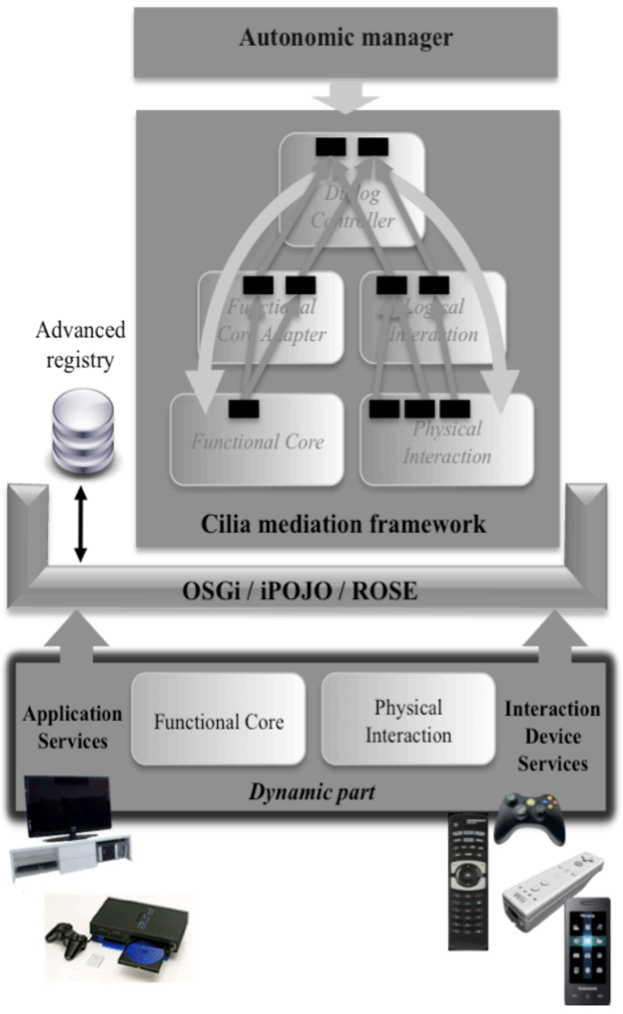


Figure 5: DynaMo overall archiecture.

As a conclusion, the DynaMo framework is built on top of readily available and proven software elements, which is required in order to attain the expected level of quality. OSGi is an industrial framework providing flexibility and service orientation on top of Java. We are using Felix, the reference open source implementation hosted by Apache. IPOJO is a service-oriented component model facilitating the development of OSGi-based applications. It is also available on Apache and is widely used today. CILIA leverages OSGi and iPOJO to provide a dynamic Enterprise Service Bus and, more generally a solution for data mediation. Cilia is also available in open source and is successfully used in collaborative projects (including industrial projects with France Telecom and Schneider Electric). Finally, ROSE is an iPOJO-based framework conceived to handle distribution in dynamic and heterogeneous environments. ROSE is made available on the OW2 forge and used in industrial settings (to provide eBooks for instance). All these software elements have been designed by the same team in the last decade and are carefully integrated.

While the two first constituents of DynaMo, the service integration platform and the mediation framework are fully described in [2], the following section focuses on the autonomic manager and its models. We articulate the presentation of the autonomic management of interaction by adopting an original point of view based on the levels of abstraction of the Arch model: Arch provides us with a structured way (i) to explain the different cases for the autonomic manager and (ii) to present a complete example.

AUTONOMIC MANAGER AND ITS MANIPULATED MODELS

Autonomic systems are made of managed artifacts and an autonomic manager. Managed artifacts are the software entities that are automatically administered by the system. Here the managed artifacts are clearly the mediation chains implementing multimodal processes. The autonomic manager is the module in charge of the runtime administration of the managed artifacts. The purpose of the DynaMo autonomic manage is to build and maintain multimodal interactions at runtime. To make its decisions, the manager uses semantics-related knowledge defined by interaction experts and contextual information provided by the execution machine. It builds multimodal interaction through the composition of predefined components conforming to the component model of Cilia presented in [2]. When a modification occurs in the environment (e.g., a new device or a new application) the manager is reactive and computes a new mediation chain or adapts the current one. Based on the dynamic capabilities of the Cilia mediation framework, the autonomic manager can manage a mediation chain at a very fine grain (i.e. the component level): when a component is replaced, the new component receives the state of the replaced one.

The manager is first driven in its decisions by high-level goals, namely policies, that can be set by the user. Moreover the user can modify the policy at runtime. Two policies are currently defined: <simple> and <bind-all>. With the <simple> policy, the goal of the manager is to bind each task of the current active application with available devices. With the <bind-all> policy, the manager starts from the devices and its goal is to bind all the devices to the tasks. This <bind-all> policy likely implies that multiple equivalent modalities (Equivalence of the CARE properties [21]) are defined for a given task.

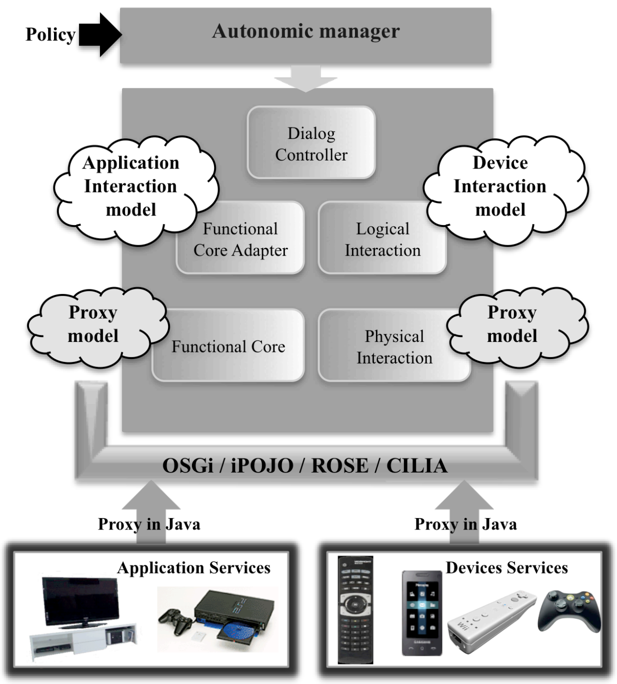


Figure 6: Models used by the autonomic manager within an ARCH software architecture.

The developers first create proxies in Java (Figure 7) and proxy models expressed in a xml language (Figure 8). The proxy meta-model is described in [2].

```
// Called by iPOJO when the instance becomes valid
private void start() {

 conn = DBusConnection.getConnection
 (DBusConnection.SESSION);

 remoteVlc=conn.getRemoteObject("org.mpris.vlc",
  "/Player",MediaPlayer.class);
}

// Called by iPOJO when the instance becomes
invalid
private void stop() {
  conn.disconnect();
}

public void playPause() {
  remoteVlc.Pause();
}

public void volume(int vol) {
  remoteVlc.VolumeSet(vol);
}
```

Figure 7: Java proxy of VLC media player (downloadable at www.videolan.org/vlc) that includes the life-cycle related code that handles the connection with VLC, and the two entry points "playPause" and "volume".

A proxy model contains information used by ROSE to track the services (devices or applications) and to start the corresponding proxies. A proxy model also contains information about the protocol (e.g., inter-process communication system D-Bus in Figure 8) as well as the ports and their types (e.g., port `playPause` and type `event` in Figure 8). Based on the proxy models, the autonomic manager is able to bind the proxies to the endpoints (i.e., an application or an interaction device) of the mediation chain. Figure 9 graphically presents the proxy models of the application VLC and the device Wiimote as endpoints of the mediation chain.

```
  <name>vlc</name>

  <discovery>
    <discriminator>org.mpris.vlc</discriminator>
    <factory>VLC</factory>
    <location>vlc-0.1.jar</location>
    <protocol>dbus</protocol>
  </discovery>

  <port>
    <codereference>playPause</codereference>
    <datatype>event</datatype>
    <direction>in</direction>
    <name>playPause</name>
  </port>
  …
</dynamo>
```

Figure 8: Excerpt of the xml proxy model of VLC: description of the communication protocol and the port playPause. VLC being an existing application, the xml proxy model describes the ports and types defined in VLC.

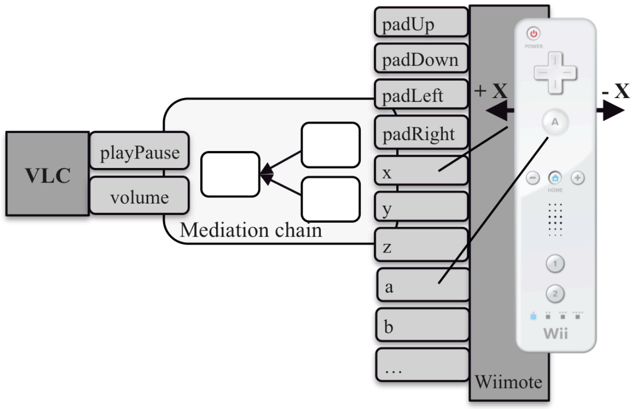
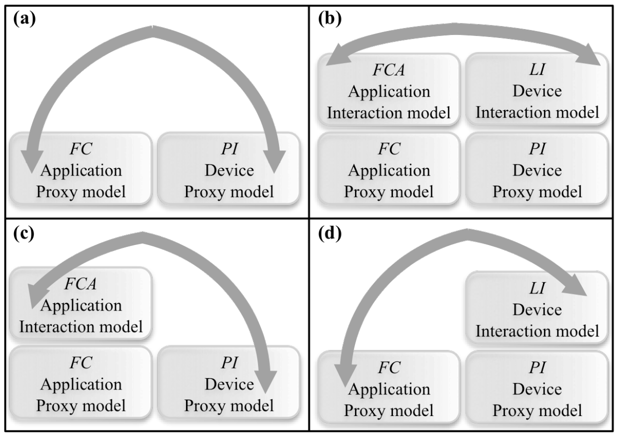


Figure 9: Excerpt of VLC and Wiimote proxy models represented as endpoints in the mediation chain.

Based on only the proxy models, the autonomic manager will generate a mediation chain from information about data type. Based on the ARCH model, this corresponds to the case where the manager directly links the Physical Interaction component to the Functional Core component as depicted in Figure 10-a. For example, the autonomic manager will bind the port `x` of the wiimote to the port `volume` of VLC, and the port 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. In this case, the autonomic manager only performs syntactic alignments to generate this mediation chain. For instance, when dealing with interaction devices providing numbers, adaptors are often necessary to align the provided values and the ones expected by the applications. In the example, the Wiimote `x` port provides values in [-180, +180] and the VLC `volume` port needs values in [0, +100], so an adaptor is introduced to perform linear transformation.

Finally in order to go beyond syntactic alignment, the manager relies on semantics-related knowledge called interaction models. An interaction model contains information about data processing, data path and data semantics. They describe the way an application and a device can be used from an interaction point of view. As opposed to proxy models, no programming skill is required for defining interaction models in a xml language. A graphical editor of interaction models is provided. An interaction model describes a partial interaction that has to be completed by the autonomic manager. An interaction model is partial because it describes only a sub-part of the transformation chain which ranges from raw data captured by devices to elementary tasks. In structuring this transformation chain along the levels of abstraction of ARCH, we defined two partial interaction models. Indeed Application interaction models and Device interaction models respectively correspond to the Functional Core Adapter component and the Logical Interaction component of ARCH (Figure 6). According to the availability of the interaction models, Figure 10-(b,c,d) depicts three cases for the autonomic manager when building a mediation chain.



(a)
FC Application Proxy model
PI Device Proxy model
(b)
FCA Application Interaction model
LI Device Interaction model
FC Application Proxy model
PI Device Proxy model
(c)
FCA Application Interaction model
FC Application Proxy model
PI Device Proxy model
(d)
LI Device Interaction model
FC Application Proxy model
PI Device Proxy model

Figure 10: Four cases for the autonomic manager according to the availability of interaction models.

For applications, the interaction model plays the role of the ARCH Functional Core Adapter as described in [7]. For instance semantic reparation can be performed by adding a new application task. By considering the VLC example, the application interaction model can add a new task to mute the volume of VLC. Figure 11 presents an excerpt of the VLC interaction model. It includes a constant generator that generates the value 0 sent to the port `volume` of VLC. Constant generator is one example of a generic function that can be used by the designer, for defining interaction models.

```
<dynamo>

 <interactionClass>mediaplayer</interactionClass>

 <proxy>vlc</proxy>

 <component>

  <baseComponent>constantGenerator</baseComponent>

    <port>
      <name>in</name>
      <connectedPort>mute</connectedPort>
    </port>

    <port>
      <name>out</name>
      <connectedPort>volume</connectedPort>
    </port>

    <property>
      <key>constant</key>
      <value>0</value>
    </property>

 </component>
…
</dynamo>
```

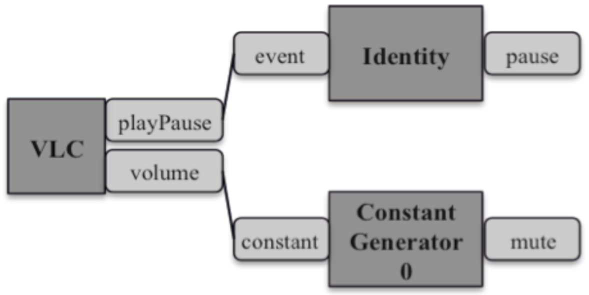
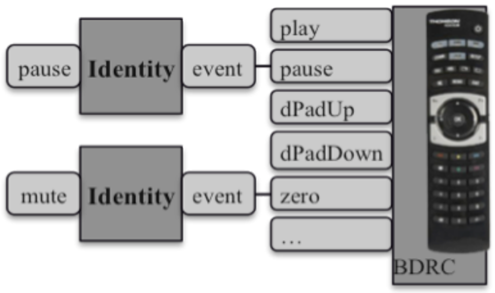


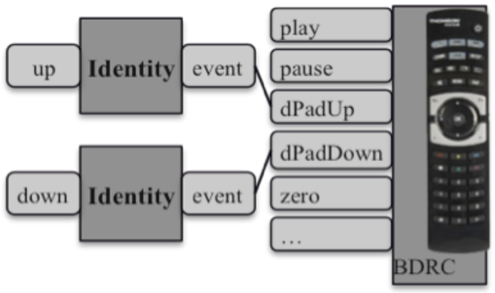
Figure 11: Excerpt of the interaction model of VLC.

For devices, the interaction model plays the role of the ARCH Logical Interaction component. It describes how to abstract events from devices. It is the right place to abstract or standardize different types of events from devices (identifier and/or parameters of a command). It depends on the syntax of the interaction language. It does not, however, depend on the semantic level of the application. As for application interaction models, the designer can use predefined processing functions within a device interaction model. For example, a triggering function sends an event as soon as it receives a value greater than a specified value. In addition generic fusion functions are provided based on the CARE properties (Redundancy and Complementarity). These generic functions are implemented by mediators developed with reuse concerns in mind and are similar to the generic components manipulated in the OpenInterface framework [23] (e.g., the Filter and Complementary generic components of Figure 1). Moreover more complex components can be used including gesture or speech recognition. The designer declares which components are to be used and bind their ports within the graphical editor. At this stage of the specification, data types can generally be ignored because the autonomic manager will be able at runtime to infer each port data type. Such inference leads to the completion of component configuration, and adds a data type converting component if necessary. For example, we have developed the multimodal map navigator as described in [23] (Figure 1) using a generic speech recognizer and a complementarity component, two mediators specific to multimodal processing. These components can be directly inserted in the interaction models, with a given configuration.

Application and device interaction models not only contain information about data processing (mediator class) and data path (bindings) but also data semantics. Indeed the autonomic manager needs to match meanings defined in the different interaction models. The current state of DynaMo supports simple semantics matching by defining interaction classes. An interaction class defines several meanings that make sense together. For example in Figure 11 and 12-a, we use the interaction class MediaPlayer that defines the two meanings *pause* and *mute*. Another interaction class is called GamePad and defines the meanings: *up*, *down*, *left* and *right*. Only one interaction class is referenced by an interaction model. The interaction designer can nevertheless define several interaction models based on different interaction classes for a given device for example, as shown in Figure 12 for the TV remote control.

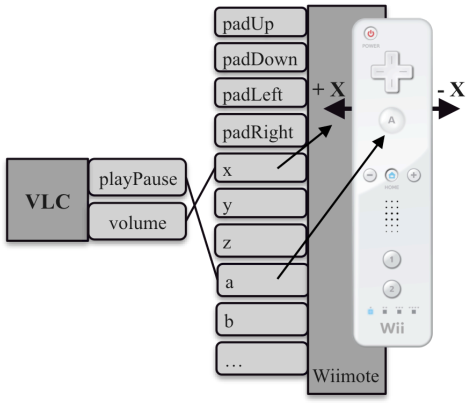


(a)

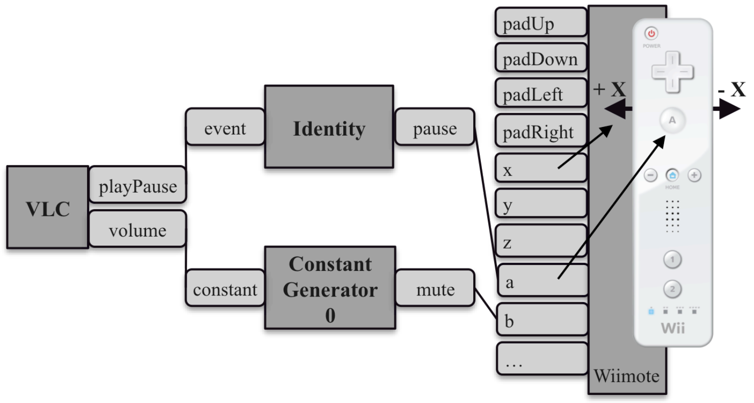


(b)

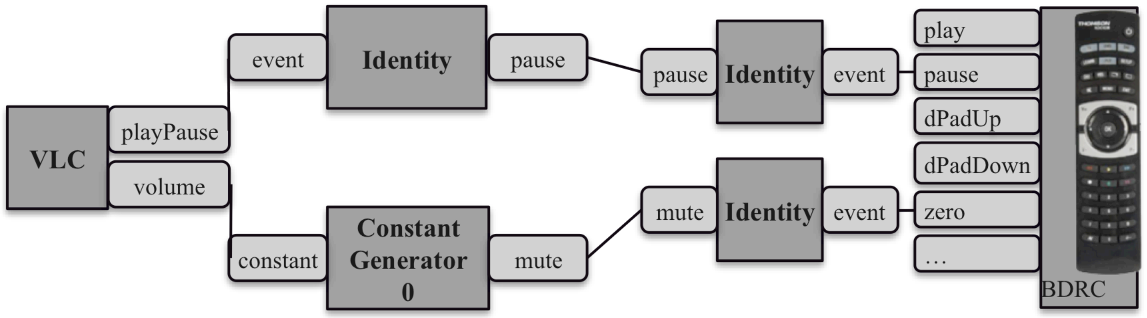
Figure 12: Excerpt of two interaction models of the BDRC TV remote control: (a) MediaPlayer interaction class (b) GamePad interaction class.



(a)



(b)



(c)

Figure 13: VLC - Generated mediation chains (a) without interaction model (b) with an application interaction model (c) with application and device interaction models.

Figure 13 illustrates how the interaction models guide the autonomic manager by considering the VLC example. Figure 13-a shows the case of Figure 10-a without interaction models. The autonomic manager only performed syntactic alignments to generate this mediation chain that links the wiimote to control VLC. Figure 13-b shows the case of Figure 10-c, with a VLC interaction model only. Since no interaction model is defined for the wiimote, the autonomic manager links port `b` of the wiimote to the new port `mute` of VLC that extends the tasks that can be performed with VLC. Finally Figure 13-c considers the case of Figure 10-b with application and device interaction models. We consider that the BDRC TV remote control has been activated. The autonomic manager receives a discovery notification about BDRC, hence it downloads the BDRC binary proxy from the repository and starts it. Amongst the interaction models of BDRC, it selects the one that uses the MediaPlayer interaction class (Figure 12-a) since VLC also has a MediaPlayer interaction model. Simple semantics matching is then performed by the autonomic manager in order to define the complete mediation chain, since the application and the device interaction models belong to the same class, namely MediaPlayer. If the interaction policy is set to <bind-all>, the wiimote will still be connected to VLC and the user can select one of the two equivalent modalities for specifying VLC tasks.

To conclude on the autonomic manager, we focus on the complementarity of modalities that imply fusion mechanisms. In DynaMo complementary modalities can be defined by using a fusion component. The complementarity generic mediator combines events close in time based on a temporal window that is a configuration parameter of the component. On the one hand, such a component can be declared within an interaction model as explained above for the example of the multimodal map navigator that combines speech with pointing gestures. On the other hand, the complementary component can be automatically added to the mediation chain by the manager.

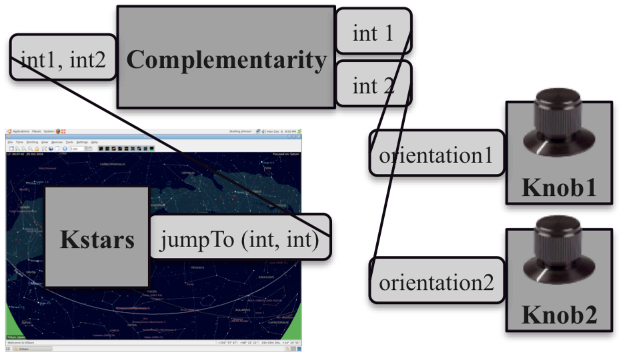


Figure 14: KStars application (downloadable at http://edu.kde.org/kstars/): Automatically generated combined usage of two knobs for selecting a point on the night sky map.

Figure 14 illustrates this case: We consider another application, KStars, that is a downloadable application that simulates the night sky, including stars and planets. The proxy of this service includes a port `jumpTo` with two integers as parameters. We consider that we have two control knobs as interaction devices for interacting with KStars. The two control knobs provide an orientation as output (the two ports `orientation1` and `orientation2`). Without interaction models, the autonomic manager will define a multimodal interaction corresponding to the combined usage of the two knobs for manipulating the night sky map. The corresponding generated mediation chain includes a complementarity component.

CONCLUSION AND RESEARCH AGENDA

The autonomic DynaMo framework is dedicated to the development and the runtime management of pervasive multimodal interaction. In pervasive environments with highly dynamic and heterogeneous applications and interaction devices, DynaMo is able to analyze and understand multiple communication means and reconfigure itself in real-time. To do so the autonomic manager uses Proxy models and Interaction models to generate and maintain a mediation chain that defines multimodal interaction techniques. The underlying execution machine then concretely realizes the mediation chain.

The clear distinction between the management of the dynamic computing infrastructure and that of multimodal interaction enables us to identify two distinct roles:

- that of the developers that define proxies so that applications and devices can be managed by DynaMo, and,
- that of the interaction designers that define interaction models at a high level of abstraction without knowing the implementation of the applications and device drivers.

Proxy models as well as interaction models guide the autonomic manager in order to define the mediation chain corresponding to the multimodal interaction processes. On the one hand, the autonomic manager has all the information to build a mediation chain: the interaction models are then complete and defined at design time. DynaMo then corresponds to the existing multimodal frameworks for which multimodal interaction is defined at design time. On the other hand, the autonomic manager is also able to build a mediation chain with incomplete interaction models. This is the case of multimodal interaction not fully defined at design time: for example a device not initially planned to be used to control a particular application is now linked to a task of this application.

DynaMo is fully operational and stable. In particular the execution machine is robust and used by our industrial partners (Schneider Electric and France Telecom especially). Its robustness and its clear distinction of concerns within its architecture enable us to build a research agenda on pervasive multimodal environments that we organize along three main axes.

Populate the DynaMo framework for experimental evaluation: We plan to add new devices and new applications to DynaMo by defining new proxies. The goal is to be able to perform experimental evaluation. DynaMo is currently studied within the Medical project (medical.imag.fr). Medical is a project that focuses on smart homes for the elderly. Its overall purpose is to provide a middleware, based on iPOJO and Cilia, allowing the development of pervasive applications. Applications are hosted by so-called "Internet boxes" and can be extended to the cloud. Interaction needs are immense in such contexts. In current settings, we are using remote control, speech commands, mouse and tactile tablets as input modalities. For instance one primary need is the configuration of applications on a tablet. While several equivalent modalities are defined at design time for configuring the applications on the tablet, it also possible to interact with applications on the tablet from far away, by using the remote control. Such interaction was not fully defined at design time. Moreover for Health applications for which values of physiological parameters must be regularly captured, multiple modalities based on different sensors (passive or active modalities) are defined. The autonomic manager may make this multimodal process of capturing values of physiological parameters more efficient, robust and safe.

Enrich the autonomic manager: We identify several research avenues to better guide the autonomic manager in its decisions. First further contextual information [8] can be used by the autonomic manager. For instance the autonomic manager could learn from the users' interaction (inputs): her/his preferences in terms of devices and/or coupling of devices with applications. This learning approach can be directly implemented in DynaMo since our architecture supports the sensing and actuating at a fine grain. Moreover situational contextual information including spatial relationships as in proxemic interactions [15] could guide the manager, for instance in selecting the devices closer to the user. A more general approach to enrich the autonomic manager and its knowledge is also to define ontologies instead of interaction classes, which correspond to very simple light-weight ontologies. As explained in [25], several ontologies for pervasive environments have been proposed. For example the European project SOFIA (Smart Objects For Intelligent Applications) explored ontology-based mechanisms for pervasive environments [19]: In this context, the Semantic Media Ontology [20] is closely related to the interaction class Media Player used in DynaMo. By using ontologies, in DynaMo, correspondences between application and device ontology entities will then be based on ontology matching mechanisms (equivalence or subsumption relations between ontology entitites).

Make the decisions observable and controllable by the user: The autonomic solution of DynaMo is appealing since autonomic adaptation does not require the user to explicitly manage her/his pervasive environment and could ideally only focus on her/his intention. Nevertheless as stated in [9], the user needs at least to understand the state of its environment and furthermore control or tune the decisions made by the autonomic manager (i.e. mixed initiative approach [17]). For these purposes a meta-UI as described in [9] must then be defined. The current meta-UI of DynaMo is basic and requires further studies: appearance and disappearance of a service (i.e., an application or a device) are notified to the user by a pop-up window and we provide partial observability of the available modalities by graphically displaying which sensors of the devices are connected to the tasks of the current active application. The ReWiRe framework [25] adopts a more advanced mixed initiative approach by displaying a "pervasive menu" that allows the users to observe and modify the current state of the pervasive environment. In DynaMo, the meta-UI to be designed must (1) make observable the current application tasks and the corresponding available modalities as well as show how to perform actions with the modalities (e.g., the symbolic 3D gestures that can be perfomed) (2) let the users modify the decisions made by the autonomic manager that could in turn learn from these modifications. It is a vast research agenda that we can explore with the current implemented architecture of DynaMo.

REFERENCES

1. A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop. *ACM SIGCHI Bulletin, 24, 1* (1992), 32-37.
2. Avouac, P-A., Lalanda, P., Nigay, L. Service-Oriented Autonomic Multimodal Interaction in a Pervasive Environment. In *Proc. ICMI 2011*, ACM Press (2011), 369-376.
3. Avouac, P-A., Lalanda, P., Nigay, L. Adaptable multimodal interfaces in pervasive environments. In *Proc. CCNC 2012*, IEEE Consumer Communications and Networking Conference, IEEE (2012).
4. Bardin, J., Lalanda, P., Escoffier, C. Towards an Automatic Integration of Heterogeneous Services and Devices. In *Proc. APSCC 2010*, IEEE Asia-Pacific Services Computing Conference, IEEE (2010), 171-178.
5. Bouchet, J., Nigay, L., Ganille, T. ICARE software components for rapidly developing multimodal interfaces. In *Proc. ICMI 2004*, ACM Press (2004), 252-258.
6. Carrino, S., Péclat, A., Mugellini, E., Omar Abou Khaled, O-A, Ingold, R. Humans and Smart Environments: A Novel Multimodal Interaction Approach. In *Proc. ICMI 2011*, ACM Press (2011), 105-112.

7. Coutaz, J., Balbo, S.Applications: a dimension space for User Interface Management Systems. In *Proc CHI 1991*, ACM Press (1991), 27-32.

8. Coutaz, J., Crowley, J., Dobson, S., Garlan, D. Context is Key. *Communications of the ACM 48, 3* (2005), 49-53.

9. Coutaz, J. Meta-User Interfaces for Ambient Spaces. In *Proc. TAMODIA 2006*, Springer (2006),1-15.

10. Demeure, A, Calvary, G., Coninx, K. CO-MET(s), A Software Architecture Style and an Interactors Toolkit for Plastic User Interfaces. *In Proc. DSVIS 2008*, LNCS 5136, Springer (2008), 225-237.

11. Dragicevic, P., and Fekete, J. D. ICON: Input Device Selection and Interaction Configuration. *Companion Proc. UIST 2002,* ACM Press (2002), 47-48.

12. Escoffier, C., Hall, R. S., Lalanda, P. iPOJO: an Extensible Service-Oriented Component Framework. In *Proc. SCC 2007*, IEEE Conference on Services Computing, IEEE (2007), 474-481.

13. Garcia, I., Morand, D., Debbabi, B., Lalanda, P., Bourret, P. A reflective framework for mediation applications. In *Proc. ARM 2011*, 10th Middleware Workshop on Adaptive and Reflective Middleware, ACM Press (2011), 22-28.

14. 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.

15. Greenberg, S., Marquardt, N., Ballendat, T., Diaz-Marino, R., Wang, M. Proxemic interactions: the new ubicomp? *ACM Interactions, 18, 1* (2011), 42-50.

16. Hoste, L., Dumas, B., Signer B. Mudra: a unified multimodal interaction framework. In *Proc. ICMI 2011*, ACM Press (2011), 97-104.

17. Horvitz, E. Principles of Mixed-Initiative User Interfaces. In *Proc CHI 1999*, ACM Press (1999), 159-166.

18. König, W. A., Rädle, R., Reiterer, H. Interactive Design of Multimodal User Interfaces - Reducing technical and visual complexity. *Springer Journal on Multimodal Interfaces, 3,3* (2010), 197-213.

19. Niezen, G., van der vlist, B., Hu, J., Feijs, L. From Events to Goals: Supporting Semantic Interaction in Smart Environments. In *Proc. ISCC 2010*, IEEE Symposium on Computers and Communications IEEE (2010), 1029-1034.

20. Niezen, G., van der vlist, B., Hu, J., Feijs, L. Using semantic transformers to enable interoperability between media devices in a ubiquitous computing environment. In *Proc. of S3E 2011*, International Workshop on Self-managing Solutions for Smart Environments at the 6th International Conference on Grid and Pervasive Computing (GPC 2011).

21. Nigay, L., Coutaz, J. Multifeature systems: the CARE properties and their impact on software design. *Multimedia Interfaces: Research and Applications, chapter 9*, AAAI Press (1997).

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

23. Serrano, M., Nigay, L. A Three-dimensional Characterization Space of Software Components for Rapidly Developing Multimodal Interfaces. In *Proc. ICMI 2008*, ACM Press (2008), 149-156.

24. Serrano, M., Nigay, L., Lawson, J-Y., Ramsay, A., Murray-Smith, R., Denef, S. The openinterface framework: a tool for multimodal interaction. *Ext. Abstracts CHI 2008*, ACM Press (2008), 3501-3506. www.oi-project.org.

25. Vanderhulst, G., Luyten, K., Coninx, K. ReWiRe: Creating Interactive Pervasive Systems that cope with Changing Environments by Rewiring. In *Proc. of IE 2008*, the 4th International Conference on Intelligent Environments, IEEE (2008), 1-8.

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

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