# A Systematic Process to Engineer Dependable Integration of Frame-based Input Devices in a Multimodal Input Chain: Application to Rehabilitation in Healthcare

Axel Carayon, Célia Martinie, Philippe Palanque, Eric Barboni, Sandra Steere

# A Systematic Process to Engineer Dependable Integration of Frame-based Input Devices in a Multimodal Input Chain: Application to Rehabilitation in Healthcare

AXEL CARAYON, University Toulouse III, France
CÉLIA MARTINIE, University Toulouse III, France
PHILIPPE PALANQUE, University Toulouse III, France and ESTIA & Centrale SupElec, France
ERIC BARBONI, University Toulouse III, France
SANDRA STEERE, Guiana Space Centre, France

Designing new input devices and associated interaction techniques is a key contribution in order to increase the bandwidth between users and interactive applications. In the field of Human-Computer Interaction, research and development services in industry and research laboratories in universities have been, since the invention of the mouse and the graphical user interface, proposing multiple contributions including the integration of multiple input devices in multimodal interaction technique. Those contributions, most of the time, are presented as prototypes or demonstrators providing evidence of the bandwidth increase through user studies. Such contributions however, do not provide any support to software developers for integrating them in real-life systems. When done, this integration is performed in a craft manner, outside required software engineering good practice. This paper proposes a systematic process to integrate novel input devices and associated interaction techniques to better support users' work. It exploits most recent interactive systems architectural model and formal model-based approaches for interactive systems supporting verification and validation when required. The paper focusses on Frame-based input devices which support gesture-based interactions and movements recognition but also addresses their multimodal use. This engineering approach is demonstrated on an interactive application in the area of rehabilitation in healthcare where dependability of interactions and applications is as critical as their usability.

CCS Concepts: • **Human-centered computing** → **User interface programming**; **Interactive systems and tool**; • **Software and its engineering**;

Additional Key Words and Phrases: Frame-based input devices, Software Architectures, transducers, interaction techniques, formal models, verification

**ACM Reference Format:**
Axel Carayon, Célia Martinie, Philippe Palanque, Eric Barboni, and Sandra Steere. 2024. A Systematic Process to Engineer Dependable Integration of Frame-based Input Devices in a Multimodal Input Chain: Application to Rehabilitation in Healthcare. *Proc. ACM Hum.-Comput. Interact.* 8, EICS, Article 259 (June 2024), 31 pages. https://doi.org/10.1145/3664633

Authors' Contact Information: Axel Carayon, axel.carayon@irit.fr, University Toulouse III, ICS - IRIT, Toulouse, France; Célia Martinie, celia.martinie@irit.fr, University Toulouse III, ICS - IRIT, Toulouse, France; Philippe Palanque, philippe.palanque@irit.fr, University Toulouse III, ICS - IRIT, Toulouse, France and ESTIA & Centrale SupElec, FlexTech Chair, Bidart, France; Eric Barboni, eric.barboni@irit.fr, University Toulouse III, ICS - IRIT, Toulouse, France; Sandra Steere, sandra.steere@cnes.fr, Guiana Space Centre, CNES, Kourou, France.


ACM 2573-0142/2024/6-ART259
https://doi.org/10.1145/3664633

## 1 INTRODUCTION

Variability in terms of users' profiles, tasks they have to perform and information they have to process to do their work, calls for new interaction means in order to increase the bandwidth (i.e. quantity of information provided to the system by the user and quantity of information perceived by the user from the system per second) inside the Human-Computer Interaction. New input devices combined in multimodal interaction techniques provide means for such a bandwidth increase (as explained in the conclusion of [37]). This demonstrates the need for extensive research and innovation in Human-Computer Interaction in order to provide bridges over this ever-widening gap between user interactions and the work users have to perform in their daily life. While prototypes and demonstrators [13], usually evaluated through user studies, are the classical mean for presenting and validating such contributions, their deployment in real-life environments is far from being frequent. We argue that this limited take-up-ability lies in the fact that these prototypes are only developed with the proof-of-concept objective in mind and providing no information or support to software developers about how to integrate them in software products (e.g. at interaction techniques [22] or at interaction services level [38] with rare exceptions such as [12] which presents how to program standard interactions using hierarchical state machines).

The more the input device and the associated interaction techniques are complex, the more difficult is the integration and finite state machines do not offer enough expressive power (not able to represent time, large number of states, data information about states, ... as demonstrated in [24]). Frame-based input devices, such as Kinect or Leap, are good examples of such innovations involving depth cameras, as they support gestures, gaze and more generally body movement-based interactions. They are, however, very different from standard input devices as they can sense up to sixty frames per second requiring high performance input chain management. Interactions made possible by such sensing capabilities are not offered by classical input devices such as mouse and keyboard. Based on these specific capabilities, they bring a large number of possibilities to interaction technique designers and to their users to perform tasks either unfeasible without them or extremely hard to perform. Research prototypes and demonstrators have been provided in multiple domains such as healthcare [19], medical imagery [34], remote control of robotic arm [35] or education [54]. Despite those clear benefits, their integration in real-life systems and environment remains seldom as demonstrated by the Microsoft™ policy (Kinect discontinued in 2015, production of Kinect for Xbox One ended in October 2017 and Azure Kinect discontinued in 2023 [3]).

We attribute this limited take-up of such promising technologies to a triple problem. First, on the interaction technique side, interactions designers proposing interaction techniques on top of these input devices do not provide concrete specifications and enough details to make it possible to interactive application developers to integrate them. Second, the problem of integrating those devices and their associated APIs into an interactive application development environment while integrating "standard" input devices is made trivial. Third, the lack of systematic process to support developers from the identification of what needs to be done (which interaction technique to develop) to the testing, validation and eventually verification of what has been developed.

This paper addresses the issue of engineering such interaction techniques exploiting the capabilities of Frame-based input devices. To this end, we propose a generic software architecture Frame-based together with a systematic process for designing these interaction techniques and for integrating them into interactive applications. This process and its associated generic software architecture allow developers to build interactive application enhanced by them. We demonstrate the use of this framework and its associated benefits (e.g. dependability of interactive applications) on a healthcare rehabilitation application involving both Kinect and Leap Motion input devices.

The paper is structured as follows. Next section introduces the interactive systems architecture MIODMIT [19] and its limitations when it comes to the integration of Frame-based input devices. Section 3 untangles the Kinect device and its associated software to demonstrate the specific needs of Frame-based input devices. Section 4 introduces a generic, systematic and stepwise process for integrating Frame-based input devices in interactive applications. The Leap Motion input device is used as a running example highlighting the common grounds with Kinect and the genericity of the issues with Frame-based input devices. Section 5 proposes a more global process for engineering interactive application embedding Frame-based input devices. Section 6 presents how to address multimodality issues arising from the integration of multiple Frame-based devices as well as the integration of Frame-based devices with "standard" input devices. Section 7 exploits a real-life healthcare application to demonstrate the applicability and the benefits of both the proposed generic architecture and the generic process for engineering interactive systems embedding Frame-based input devices. Section 8 positions the contributions with respect to related work, while section 9 concludes the paper and highlights paths for future work.

## 2 RELATED WORK

The integration and use of Frame-based devices is a large topic being addressed in numerous papers and has been applied to multiple domains. However, when it comes to the engineering of interactive application integrating Frame-based device, the list of papers is rather short and can be regrouped into two main categories. First, papers proposing tools or methods to make it easier to use and develop applications integrating one or several independent Frame-based devices. Second, papers addressing the issue of integrating multiple Frame-based devices together in a multimodal way. Beyond this work, research contributions have been proposed to address the software and hardware architectures of interactive applications offering Frame-based devices. These contributions are reviewed in section 2.3.

### 2.1 Contributions to the development applications integrating independent Frame-based devices

Teddy Seyed et al [50] proposed a toolkit for prototyping and developing multi-sensor and multi-device environment, with a focus on spatial interactions. This toolkit integrates inputs from Leap Motion, Kinect v1 and v2, Apple iBeacon and Mobile device sensors. However, the architecture they propose is specific to this exact set of devices, is not extensible and does not provide support to integrate other devices. The toolkit they propose supports the development of applications in a closed set of integrated development environments and doesn't provide any explicit support for developing new gesture-based interactions (than the ones identified). Sluÿters et al. proposed QuantumLeap [51], a framework that supports the engineering of GUI applications with a Leap Motion device. The framework focuses on gesture-based interactions only and is extensible, allowing to change part of the architecture to meet the application's needs and even to change the device. However, it is limited to JavaScript only and although it can support multiple devices, the developer is not helped for producing the code for the missing parts and to integrate then in the framework. Nebeling et al. proposed XDKinect [40], a toolkit for cross device interaction using Kinect. They highlight the relevance of integrating multiple interactive devices, Frame-based and others, to support full-body interactions. The toolkit (and its associated architecture) enables the use of distributed clients, enabling interactions with a Kinect not directly connected to the client computer. However, XDKinect natively only supports the integration of one Kinect. Pedersoli et al. proposed XKin [44], a framework for designing interactive applications for American Sign Language recognition with Kinect in C/C++. This framework is dedicated to a specific user task, a specific interaction language and proposes a specific architecture for this application. Finally,

Terven et al. [52] proposed KinZ a framework dedicated to the Kinect Azure in Python and Matlab. All of those papers acknowledge the difficulty of engineering applications that use Frame-based devices. They propose means to ease and support their development but all those solutions are bound either to specific devices, or programming environments and offer limited extensibility.

### 2.2 Contributions to the use of multiple Frame-based devices

Since different types of Frame-based devices offer different interaction possibilities, several contributions focus on using multiple Frame-based devices together and to integrate them to allow to:

- increase usability through multimodality,
- improve their accuracy through redundancy,
- increase reliability in case of faults in the devices.

Penelle et al. [45] proposed an approach to fuse Kinect v1 and Leap Motion data to improve tracking accuracy. Li et al. [34] proposed a similar approach but with Kinect v2 and Leap Motion, and applied it to the manipulation of objects in a 3D unity scene. Wu et al. [56] first proposed a setup integrating three Kinect v2 to provide reliable real-time full-body tracking in VR 3D environments. Wu et al. [58] later refined this approach by adding a fourth Kinect and a Leap Motion to reliably track (in a 3D environment) not only the users' positions but also their orientation and hands positions. Finally, Wu et al. [57] used this setup to study its impact on communication between multiple users in a VR environment. All of those approaches show how reliability of tracking can be improved using diversity (according to dependability computing terminology [49]) of redundant Frame-based devices. Other papers focused more specifically on the different types of interactions and how to accurately identify them. For instance, Kumar et Al [30] [29], proposed two different approaches based on Hidden Markov Models to accurately track American Sign Language in the dynamic pose by fusing Kinect and Leap motion data. Marin et al. [36] first proposed an approach to recognize American Sign Language in static poses using a Kinect together with a Leap motion and then refined their approach with the same setup to detect different gestures in dynamic poses.

While those papers demonstrate very promising results and explain how multiple Frame-based devices can be exploited to accurately track the human body and to recognize different types of interaction, they always focus on the algorithm which were built in a craft manner. They do not provide a systematic way to integrate the Frame-based input devices, meaning that any developer that needs to integrate Frame-based input devices has to adapt their algorithm to the target context and development environment without methodological or technological support.

### 2.3 Harware+Software architectures addressing Frame-based input devices

When contributions are considering the software engineering of interactive applications embedding non-standard devices offering multimodal interactions, architectural models are proposed (e.g. see [53], [27]) as a way of describing and explaining the various components involved. More generic ones have also been presented, but they are usually bound to one type of modality such as touch interaction [20] or speech interaction [28]. As far as interaction techniques are concerned, dedicated software architectures have been proposed too but focusing on a specific problem raised by a specific kind of interaction technique in an interactive application. For instance, [41] presents the Accelerated Touch Architecture and [23] the Layered Multi-touch Architecture but both only address specific problems related to touch input. MUDRA [25] is an outsider contribution which proposes a framework embedding a generic architecture for multimodal interaction. However, MUDRA architecture only addresses a defined set of input devices and does not provide a generic approach making explicit how new devices can be instantiated. Beyond, the architecture does

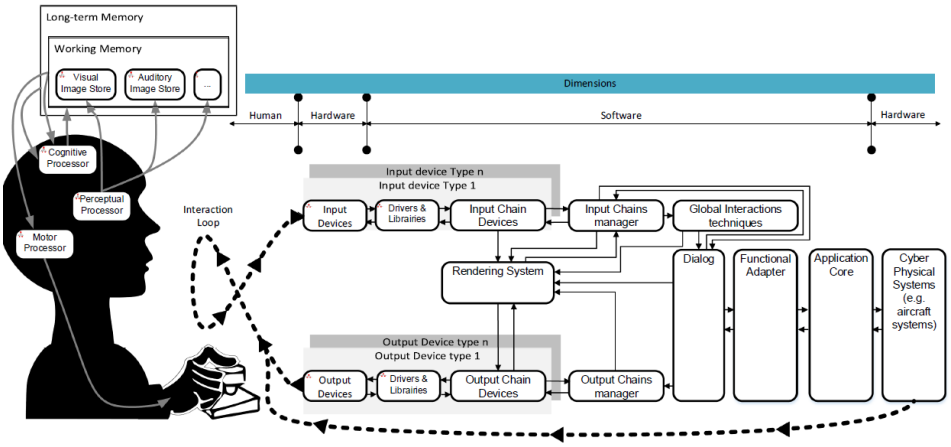


Fig. 1. Overview of the MIODMIT architecture (from [17] refined from [19])

not address output i.e. how information is presented and how output devices are exploited. The MIODMIT [19] architectural model is one of the latest attempts to propose a generic way to describe any kind of interactive systems taking into account input and output evenly. We selected it because it is the only one that enables us to describe in a precise way the main components, hardware and software, of the input and output chains of the interaction between a user and an interactive system.

*2.3.1 MIODMIT architecture and process.* Figure 1 presents MIODMIT architectural view of an interactive system, from the input and output devices (left side – connected to the human) to the application core (right side). Rounded boxes represent functions or components providing functions and arrows represent information and control flow.

*2.3.2 The specificities of the interaction (input and output).* The top left side of the architecture on Figure 1 corresponds to the processing of the input devices including device drivers and associated interaction techniques that are needed for interacting with the application. Interaction techniques have a tremendous impact on operator performance and their design and tuning is key. Even standard interaction techniques encompass complex mechanisms (e.g., modification of the cursor's movement on the screen according to the acceleration of the physical mouse on the desk). MIODMIT describes that interaction mainly takes place through the manipulation of input devices (e.g., keyboard or mouse) and the perception of information from the output devices (e.g., a computer screen or speaker).

The grayed-out boxes labelled "Input Device Type" handle events flow for each type of input device made available to the user. For instance, having two mice and a voice-recognition system would require two separate "Input Device Type" grey boxes as they require very different input processing (continuous flow of sound versus discrete events) flow of sounds and don't operate in the same time frame (much more time is needed to process speech). Following the normal flow of events (in which the interactive system is idle waiting for input from users) a given "input device" sends events to the "Driver & Libraries". The "input chain device type 1" transforms the raw data into higher-level information (e.g. transformation of the amount of motion of a mouse (dx, dy) into absolute coordinates for the mouse pointer). Such information is then processed by the "input chains manager" (e.g. picking function connecting the input event to user interface objects) that possibly fuses information from the various input devices types.

The same holds for the output processing which is a mirror of the input side. The "rendering system" component in the middle of the diagram includes an immediate feedback function and more sophisticated state-based rendering functions (e.g. splitting information to be presented of various output devices).

*2.3.3 The dialog and the application core.* In Figure 1, the right side of the Software section of the architecture (from the Dialog box to the Application core box) corresponds to what is usually called interactive applications. Fused information incoming from the "input chain manager" is dispatched either directly to the "Dialog" or to "Global Interaction Technique" which behaves as a transducer as defined in [6] and then dispatch information to the "Dialog". Both the "Dialog" and "Application Core" systems have a similar responsibility as in standard interactive architecture models such as Seeheim [46] or ARCH [31]. The "Functional Adapter" have a similar responsibility as the "Functional Core Adaptor" in ARCH [31].

### 2.4 Limitations for Frame-based input device management: tackling the myth of the different input device types

According to MIODMIT architecture [19], each type of input device chain is processed by three distinct components: the Input Device itself which communicates with the Driver and Libraries which in turn communicates with the Input Chain Devices (which involves managing and combining information from the same type of devices). As they are mature and well mastered "traditional" input devices (such as mice, trackballs or joysticks) tend to share the same functions, events and data structure, meaning that different mice can be managed in the same way and are thus gathered in a single chain (called input device type).

On the opposite, Frame-based devices, even from the same product range (e.g. Kinect v1 and Kinect v2), that share similar features can have very different characteristics with a different workflow, different API and compatible OS, as shown in [55] comparing Kinect v1 and Kinect v2 (see Figure 2). Frame-based While both devices share the same purposes and similar internal hardware cameras, they provide different results with different drivers and different data structures. Thus, unless using multiple times the same device as proposed in [56], engineering an interactive system exploiting multiple Frame-based devices requires to integrate them as different types of devices. This is different from the original proposal of MIODMIT and has been overlooked by the authors.

## 3 A CONCRETE EXAMPLE OF INTEGRATING A KINECT V2: A FRAME-BASED INPUT DEVICE

This section presents the steps that have to be performed in order to integrate a Kinect Frame-based device into an interactive system. While presented on the Kinect v2, they are the same for any type of Frame-based device (as demonstrated in the case study section).

### 3.1 Integrating the hardware

The first step is to connect the hardware with the platform that will run it. This means looking at the specification and being sure that the system can be physically plugged into and recognized. This is especially important when using multiple devices because not only they might require multiple connection ports but they may also interfere with other devices already connected. One simple example is when plugging two mice on Windows which will merge input and both mice will be associated to the same pointer on the screen.

For Kinect v2, this means that only one USB A port is needed as shown on Figure 3.

**Table 1** Comparison between Kinect v1 and Kinect v2

| | | Kinect for windows v1 | Kinect for windows v2 |
|---|---|---|---|
| Color | Resolution | 640× 480 | 1920× 1080 |
| | fps | 30fps | 30fps |
| Depth | Resolution | 640× 480 | 512× 424 |
| | fps | 30fps | 30fps |
| Sensor | | Structured light | Time of flight |
| Range | | 1.2 ~ 3.5m | 0.5 ~ 4.5m |
| Joint | | 20 joint / people | 25 joint / people |
| Hand state | | Open / closed | Open / closed / Lasso |
| Number of Apps | | Single | Multiple |
| Body Tracking | | 2 people | 6 people |
| Body Index | | 6 people | 6 people |
| Angle of View | Horizontal | 62 degree | 70 degree |
| | Vertical | 48.6 degree | 60 degree |
| Tilt Motor | | Yes | No |
| Aspect Ratio | | 4:3 | 6:5 |
| Supported OS | | Win 7, Win 8 | Win 8 |
| USB Standard | | 2.0 | 3.0 |

Fig. 2. Comparison of Kinect v1 and Kinect v2 from Cai et al. [15]

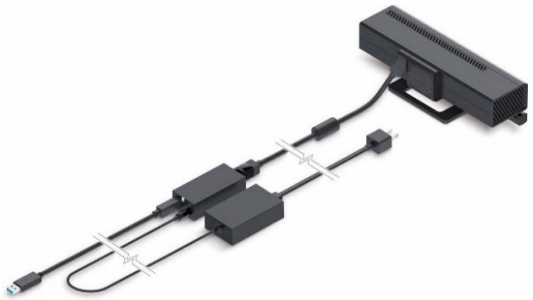

Fig. 3. Overview of the Kinect v2 connector

### 3.2 Exhaustive understanding of the input device and description of its capabilities

This step requires performing a thorough analysis of the driver and the API which are provided with the input device. For simple input devices such as the mouse, this is rather simple as presented in [6]. For a Frame-based device which embeds numerous sensors, the list is long and complex. In order to untangle this, we propose to use the CORBA Component Model [42] (CCM) which makes explicit the facets of the component (which methods it calls), the receptacles (which methods it offers) and event sink (events it is listening to) and event sources (events it produces). We use CCM due to its readability and its adequation to describe event-based components. We don't use CORBA-based middleware as a broker for runtime of interactive systems execution due to its low performance (critical when handling 30 images per second with Frame-based devices). This limitation has been demonstrated in [8]. However, researchers in [9] and [10] have used the same ICO notation which is used for describing the behavior of CORBA services.

Figure 4 presents the exhaustive CORBA Component Model of the Kinect v2 based on its official API [2]. Building this model is time consuming but is very useful to understand both the capabilities of the device and how to integrate it.

You may notice uppercase/lowercase inconsistencies in the naming of methods in Figure 4, this it is not a mistake and it is the actual name of the methods found in the official Kinect documentation [1].

The following methods are provided by the API to initialize the device:

- GetDefault(), which returns an instance of the software object corresponding to the plugged in Kinect
- Open() which opens a connection with the Kinect
- get_IsAvailable() is used to detect if the Kinect is available,
- get_KinectCapabilities() gives constants about Kinect capabilities like if it support audio, facial recognition ...,
- get_UniqueKinectID() give the unique ID of the Kinect.
- The API also offers a set of events produced by the Kinect
- IsAvailableChanged is an event triggered when the Kinect becomes available.
- PropertyChanged which is a required event in CCM but not exploited by the Kinect

In order to exploit the sensors of the Kinect, the API provides also a set of methods and events. The Kinect provides seven different kinds of frames, an RGB Frame, an Infrared Frame, a Long exposure Infrared Frame, a Depth Frame, an Audio Frame, a Body Index frame and a Body Frame. These frames are all represented in Figure 4 as components inside the container called KinectSensor. These frames contain information which may be accessed by means of a get_XXXFrameSource method (where XXX is the name of the frame).

For readability purpose, only the body frame is detailed but they all share the same structure, offer the same methods and trigger the same events (with different data inside). The Body Frame source component triggers only two events and is not a sink.

These events are:

- FrameCaptured event which is raised when a frame is captured by the sensors
- PropertyChanged which is a required event in CCM but not exploited by the Kinect

The Body Frame source provides the following methods:

- SubscribeFrameCaptured() used to subscribe to the event FrameCaptured
- UnsubscribeFrameCaptured() used to unsubscribe to the FrameCaptured
- get_BodyCount() gives the max number of bodies that the Kinect can track (6),
- get_IsActive() returns if the bodyFrameSource is active,
- get_KinectSensor() give the main sensor component,
- overrideHandTracking(int id) allows to change the Kinect hand tracking prioritization, prioritizing hand tracking on the body with the id "id" given in parameters
- overrideAndReplaceHandTracking(int old, int new) allows to change the Kinect hand tracking prioritization, removing prioritization on the body with the id "old" and prioritizing hand tracking on the body with the id "new" given in parameters
- getCoordinateMapper() method, which is a mapper that allows to map point from one point to another, which allows for example to known where is placed a body point on the RGB image.

This step is very important because it highlight all the elements that the device provides. While not all the methods and events might be needed, this step will ensure to understand all the capabilities of the device. This means that without this, developers will not only have a poor understanding of the device, but will potentially re-implement data or methods that are already provided.

### 3.3 Preparing the input device for connecting it with the programming environment

This section presents how to interface (in the software engineering meaning) your interactive application with the driver of the input device. This task is trivial if the driver provided by the manufacturers is written in the same programming language as the one of the interactive application.

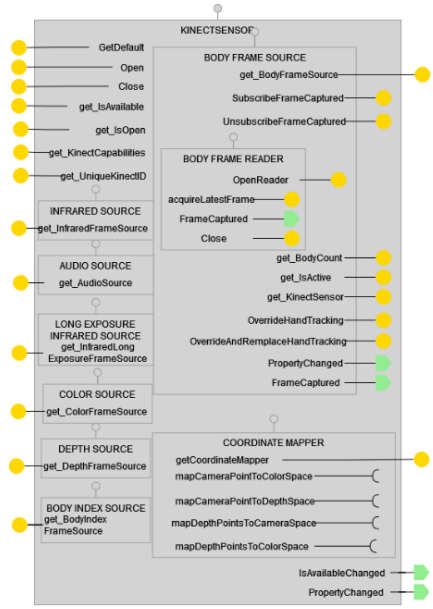


Fig. 4. Component Model of Kinect V2 API (driver)

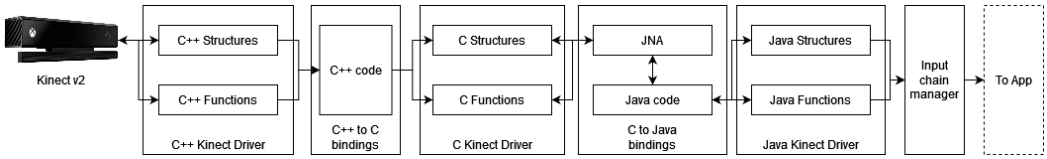


Fig. 5. Overview of the bindings pipeline from C++ driver to Java

If not, there is a need to produce a specific software entity usually called “bindings” to make the connection possible.

Bindings are used to use a library written in one language in another language. Because a lot of libraries are written in low-level programming languages like C and C++, other languages, especially high-level ones, do provide ways to call those languages. In the worst-case scenario, there might not be a direct mapping like our case with the Kinect, meaning it means that you need to write another binding in an intermediate language.

Finally, because bindings are just translating one language to another, writing them involves a relatively low algorithmic complexity. However, it requires the developer to have a deep understanding of how both languages work and their data structures to keep the semantics exact while translating from one language to another. The component model produced in the previous step is key to ensure the exhaustivity of this translation.

Kinect v2 API is provided for three languages: C++, C# and Visual Basic .NET. If Java is the programming language used to program the interactive application there will be a problem as Java can call C libraries through Java Native Access [21], but not C++, C# or Visual Basic .NET libraries. This means that two bindings have to be written: one in C++ to convert all the functions to C functions and a second one in Java to convert all the C functions to Java functions. This process is presented in Figure 5. The official API provided by the constructor is written in C++ which means

we have to call C++ methods in order to communicate with the device. Java does not allow methods call written in C++. However, C methods can be called. There is thus a need to produce a mapping written in C to access C++ methods. It is important to note that this is a specificity of the Java language. Depending of the target language of the application, such a in-between language might not be needed. For example, bindings to call C++ code are available in Python.

Bindings were absent from the original MIODMIT architecture and need to be added. The revised architecture is presented in section 3.5.

### 3.4 Producing a reliable transducer for the Input Chain Device component

In order for the interactive application to exploit events produced by a) the input device, b) its driver and c) possibly its bindings, the information produced must be organized into what is called a transducer. The term transducer was originally introduced by Bill Buxton [14] and formally defined in [6]. A transducer is software entity that converts an input stream of events into an output stream. This conversion may be the result of a process which may be complex. In the case of a mouse input device, a transducer would produce Click or DoubleClick events from sequences of press and release of a mouse button. This component is complex to produce as its internal behavior may be rapidly complex as demonstrated in [5] for the keyboard, in [6] for the mouse or in [24] for multi-touch devices.

Transducers may exhibit various desired or undesired properties as introduced in [5] where they identify three generic properties: Chatty, Regular and Sly. These properties are related to their behavior in terms of event sink and event source. The authors recommend transducers to be at least Regular meaning that every event they receive from the driver is forwarded to the application level. While this is true for "standard" input devices, it might not be the case for Frame-based input devices. Indeed, these devices embed cameras which produce large streams of data that might not be relevant for the interactive application. In that case a Sly transducer might increase the overall performance of the interactive application.

As an example, we present here a generic, regular transducer which forwards all the events it receives. Tuning this transducer to the needs of the interactive application is key as demonstrated in the case study section.

For the description of the behavior of the transducer, we decided to use Interactive Cooperative Objects (ICO)[39], because:

- They offer the explicit representation of states, events and methods;
- They handle large state space thanks to the underlying Petri nets;
- Several transducers have been presented, so it is easier to produce new ones;
- The formal model makes it possible to verify properties on the transducer;
- They bridge the gap between modeling and implementation reducing workload for developers using the CIRCUSsuite[16].

It is important to note that ICOs is only a mean here and not a goal. We use it because it provides the benefits presented above but developers may implement the behaviors of transducers with the programming language of their choice. However, in that case, testing and debugging the behavior of the transducers will be a complex, unsupported task.

Figure 6 presents an overview of the ICO model of a generic, regular transducer of the Kinect V2. On that model there are eight blocks. The initialization part describes the behavior in charge of the connection to the Kinect while each of the seven other ones corresponds to the behavior to handle one of the seven Frames produced by the Kinect.

Figure 7 present an extract of an ICO model of Figure 6 describing the behavior of the part of the generic transducer that handle the initialization of the Kinect. In the ICO notation, tokens stored

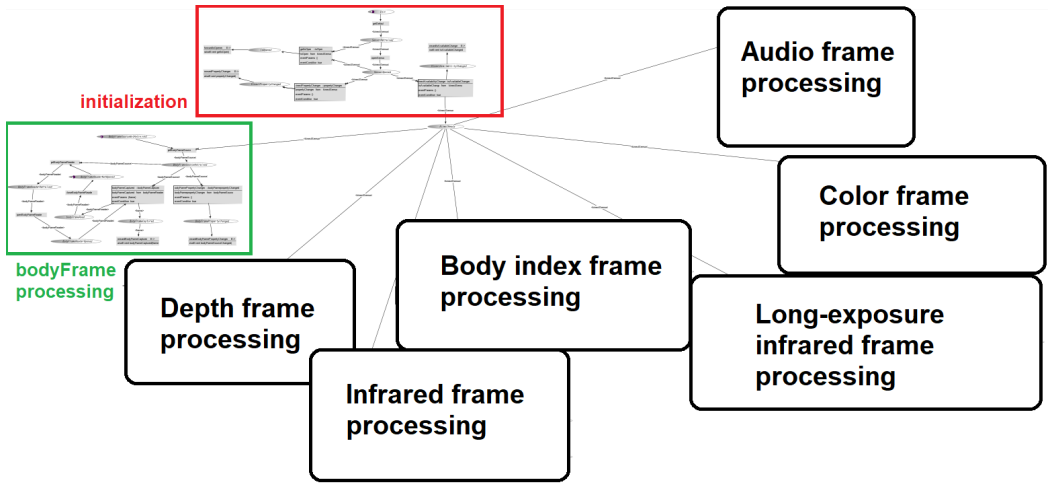


Fig. 6. Overview of the ICO model of the generic Kinect transducer- all the images are available in high resolution and high quality here https://sites.google.com/view/eics2024beyondprototyping/

in the places can hold a typed set of values and can be references to other objects. The values of the tokens can be used in preconditions of transitions to restrict their fireability. The references make it possible to call methods on these objects inside the transitions. For instance, in Figure 7 the token in place "SensorRetreived" may hold a "kinectSensor" which is the default sensor instance used by the Kinect driver to communicate with the device. The ICO formalism allows the use of test arcs that are used for testing the presence and values of tokens without removing them from the input place during the firing of the transition (e.g., the arc connecting place 'SensorOpened' to transition 'kinectAvailabilityChanged' in Figure 7 is a test arc). The ICO formalism also supports multicast asynchronous communication (event-based communication). Events may be associated to a transition meaning that the transition will only become fireable upon the reception of that event. For instance, the transition 'kinectAvailabilityChanged' (on the right-hand side of Figure 7) is fired upon the receipt of an event named isAvailableChanged. The firing of this transition deposits of a new token in place "KinectReady" and in "KinectAvailabilityChanged". It does not consume the token in place "SensorOpened" as the connection is made by a test arc. The place KinectReady is an input place of all the other parts of the model meaning that the processing of frames can only start after the initialization.

In a nutshell, Figure 7 describes the behaviour that, in order to get connect to the Kinect, you must first get the default sensor, open it, wait for its availability and, at last, access it.

Figure 8 shows an extract of the transducer ICO model handling the Body Frame (which is computed information containing detected bodies by the Kinect). The behaviour is as follows:

- First acquire from the sensor a "bodyFrameSource"
- Send event bodyFramePropertyChanged() in the code of the transition called forward-BodyFramePropertyChanged if its properties changed.
- For the processing of frames:
  - Acquire a new "bodyFrameReader" (transition getBodyFrameReader),
  - Open the "bodyFrameReader" (transition openBodyFrameReader)
  - Wait for the reception of an event "bodyFrameCaptured"

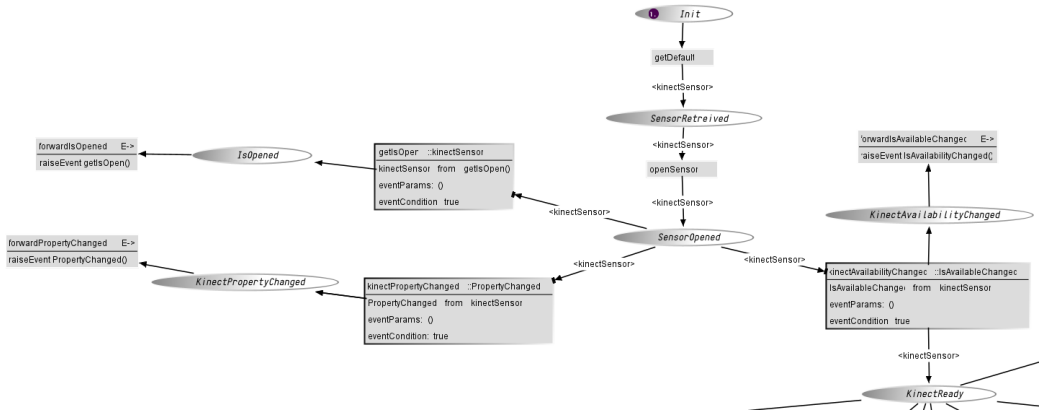


Fig. 7. Extract of the ICO model of the generic transducer initializing Kinect v2 connection

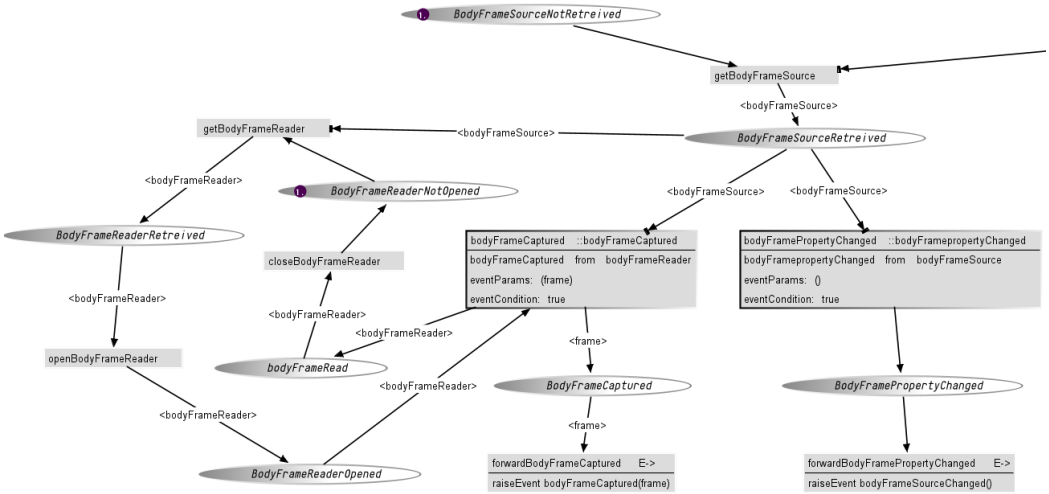


Fig. 8. Extract of the ICO model of the general transducer handling Kinect v2 body frame

- Send event “bodyFrameCaptured” (see instruction raiseEvent bodyFrameCaptured(frame) in transition forwardBodyFraneCaptured at the bottom-center of Figure 8).
- Close the bodyFrameReader to be able to the process the next frame (transition close-BodyFrameReader).

The processing of the other six frames provided by the Kinect is exactly the same. This is graphically visible on the ICO model of Figure 6. Checking this equivalence on source code is more complex.

### 3.5 MIODMIT V2- a revision integrating specificities of Frame-based devices

In order to take into account the information above, we propose here a small revision of MIODMIT architecture. As this paper focusses only on input devices, only the upper left part of the model is impacted. Specific chains should be added to each Frame-based input device but also for each

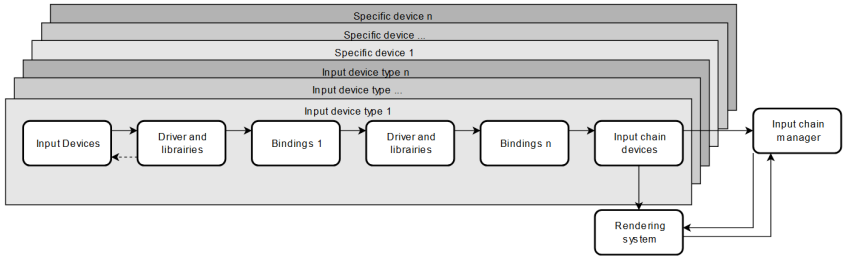


Fig. 9. Revised version of the MIODMIT architecture

device which cannot be associated to a device type. Beyond, the production of one or more bindings in a given programming language is made explicit.

Finally, we also have removed the arrows from the input chain manager to the drivers and libraries as they are not relevant. However, the dotted line between driver and libraries represents, for instance, the polling of data by the driver inside the hardware of the input device.

## 4 A GENERIC PROCESS FOR INTEGRATING FRAME-BASED INPUT DEVICES

This section presents a generic process for integrating Frame-based input devices into interactive systems. We first present the process and then show its application to the integration of a Leap Motion. This is similar to what was informally presented when integrating the Kinect v2 in section 3.

### 4.1 Generic process

The process starts by getting the actual input device and its documentation and study in order to get the exhaustive list of services, attributes and events it can produce. During this understanding phase, a component model of the device is produced and refined. In order to know if there is a need to produce or not a binding between the component of the device and the interactive application, there is a need to know what is the target programming environment and the target language of the interactive application that will exploit the device. If the target programming language is the same as the driver language, this phase is skipped and no driver is produced.

Next step is to produce a generic transducer for the device. This transducer adds behavior to the structural description of the component. This behavioral part is key to produce components that plug and play as demonstrated form CORBA components in [11]. More precisely, the transducer should forward to the Input Chain Manager all the events it receives from the device driver. Beyond, it should offer to the Input Chain Manager all the methods it can call on the device driver. It should also encompass the constraints on these methods which are typically depending on the state of the driver (e.g. cannot call a method close on a driver in a closed state). After that we can do validation and testing to demonstrate that the generic transducer matches the offered services and events from the device. If not, we need to re-work either the binding and the generic transducer until it matches what is provided by the device.

### 4.2 Application of the process to the Leap™ Motion

To show the applicability and usefulness of the process, we apply it for the integration of a Leap Motion device [4] to an interactive application developed in Java. The Leap Motion is a Frame-based input device dedicated to the capture of hands and hands movements in real time.

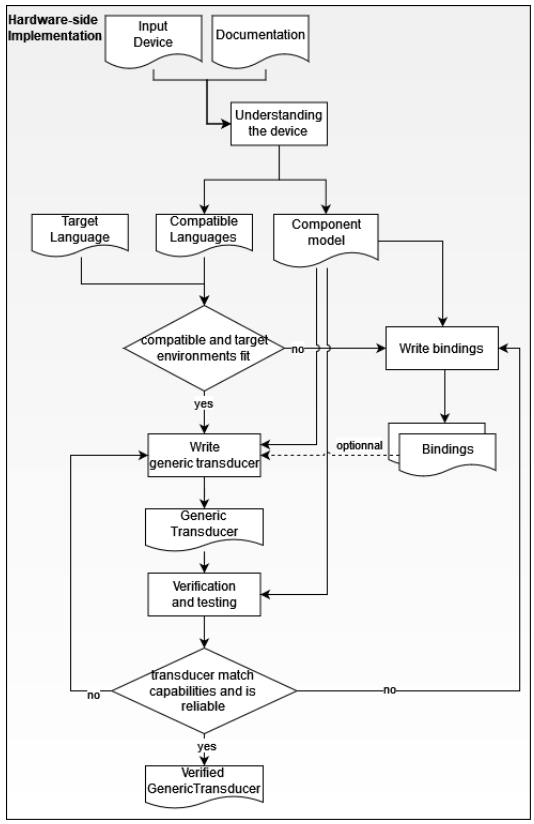


Fig. 10. Generic process for Frame-based input from the device to the generic transducer

First, we need to understand how this device works by accessing the official documentation. From there we learn that only two languages are supported by the driver: C# and C languages[1] (legacy bindings also exists for C++, Java, JavaScript, Python, and Objective–C but are no longer supported)[1]. We also get from the documentation that the Leap is producing a set of events and offering a set of methods. They are represented in the component model of Figure 11.

As the target language for the interactive application is Java, we need to write a binding from C to Java using JNA that implements the same methods and events as the one presented in the documentation, the same way we did for the Kinect in section 3.3.

JNA is indeed usually slower that JNI, but has the added benefit of being much easier and faster to write and implement. Because our testings have found that we can already reach a steady 30frames per seconds using JNA, which is the maximum framerate of the Kinect, we choose valued JNA over JNI in order to save development time. However with others devices who are more inect f intensive, JNI might be used instead if needed to reach the desired performance.

Due to space constraints, we do not present here the entire model of the generic transducer of the Leap. Instead, Figure 12 shows an excerpt of that ICO model. The workflow is different from the one of the Kinect V2 presented above as for that device the communication is event-based. This is easily seen on the component model of Figure 11 which only feature event sinks and event sources

[1]Ultraleap Hand Tracking Overview. Retrieved February 9, 2024 from https://docs.ultraleap.com/hand-tracking/

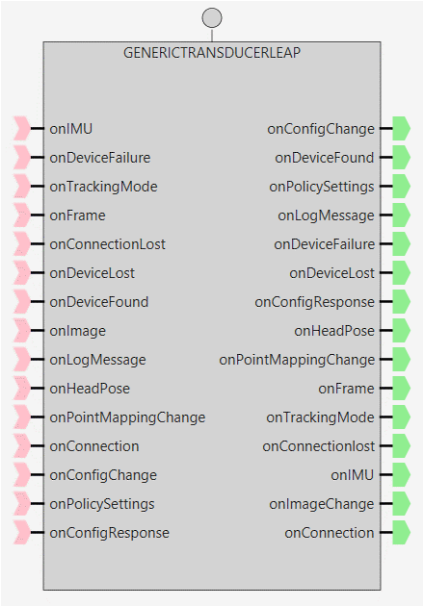


Fig. 11. CCM component model of the Leap

(according to CCM terminology). We then need to represent the states and the evolution of states when events are received and in which states the events are produced.

At initialization, after opening the connection, we need to wait for a device to be found. When this is done the driver needs an allocator to allocate data in the memory, then initializing the policy settings of the Leap which is then connected and running. More details about the processing of events by the transducer of the Leap will be presented on the case study together with the verification activity. Beyond, we will present how the generic transducer is transformed into a specific one to meet the needs of the application.

## 5 ENGINEERING INTERACTIVE APPLICATIONS EMBEDDING NEW INPUT DEVICES

When building interactive applications, several input devices may have to be integrated. This section presents a stepwise process to integrate several input devices, whether they are frame based or not, in an interactive application.

### 5.1 Stepwise process for the design and development of interactive applications requiring specific input devices

Figure 13 presents the main steps of the generic process for integrating input devices in a multimodal input chain. It decomposes in six main steps, each of them being presented in the following sub-sections. First step is the prototyping of the interactive application and is presented in section 5.2.1. Second step is to identify the input devices and is presented in section 5.2.2. The third step is to integrate a Frame-based input device for each Frame-based input device needed (section 5.2.3), which produces a generic transducer (offered services and behavior description) for each integrated Frame-based input device. The fourth step is to produce specific transducers for the input devices according to the requirements for the interactive application (section 5.2.4). The fifth and last step is to combine modalities in the input chain manager (section 5.2.5).

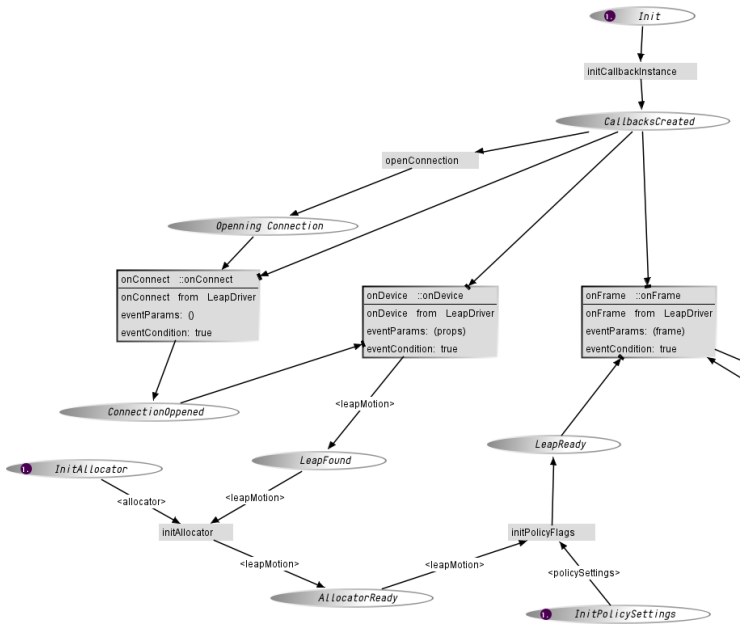


Fig. 12. Extract of the ICO model of the generic transducer for the Leap Motion Controller (initialization)

## 5.2 Description of each step of the process

*5.2.1 Prototyping.* The first step is the prototyping of the interactive application. It takes as input the user needs and requirements and is meant to be applied through user-centered design practices. This step outputs a defined list of required interaction techniques, along with a list of generic devices and selected modalities that will support the interaction techniques.

*5.2.2 Identify input devices.* The second step is to identify the concrete input devices that implement the characteristics of the required generic input devices and that will enable the implementation of the required interaction techniques. The capabilities and characteristics of the concrete input devices are checked to ensure that they match the required modalities and interaction techniques. The output of this step is a list of concrete input devices that match required modalities and interaction techniques.

*5.2.3 Integrate Frame-based input devices.* The third step is to integrate each Frame-based input device following the generic process presented in section 3. The output of this step is an exhaustive and detailed list of the service offered by each Frame-based input device (which correspond to the component model produced as shown in Section 3.2), as well as the description of the behavior (ICO model) of the generic transducer for each Frame-based input device.

*5.2.4 Produce specific transducers.* The fourth step is to produce specific transducers for each Frame-based input device. Each generic transducer is reviewed to identify the relevant events and methods for the required interaction techniques. From this set of relevant events and methods and using the generic transducer, a new transducer is produced and tune. The new transducer is a modified version of the generic transducer in which the events and methods that are not required have to be filtered out. This new version of the transducer, which contains only the relevant events

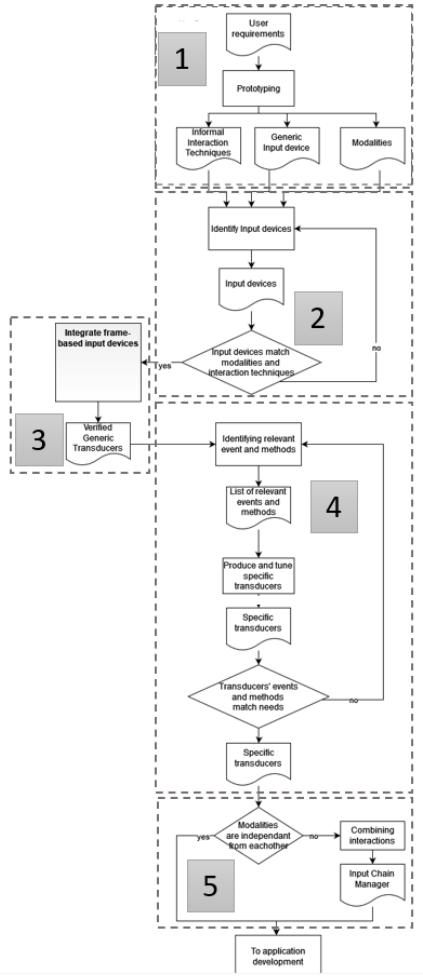


Fig. 13. Generic process for integrating input devices in a multimodal input chain

and methods is called the specific transducer. This step of the process outputs a specific transducer for each required Frame-based input device.

While the production of specific transducer is not a process specific for Frame-based device, they produce a continuous stream of image, which is way more data than what is produced by "traditional devices" such as mouse or keyboard who only produces a few bytes of information. Thus, for traditional devices, making a regular or chatty transducer was the way to go because it would not make any impact on the overall performance of the system. In the particular case of Frame-based device, regular or chatty device can lead to performance issue by processing a lot of unnecessary data. Thus, Frame-based device is a special case where sly transducers are tolerated for performance reasons.

*5.2.5 Combine modalities in the input chain manager.* The fifth step of the process is to combine the modalities if relevant with the required interaction techniques. Two possibilities have to be handled: multimodality with several Frame-based input devices and multimodality with Frame-based input

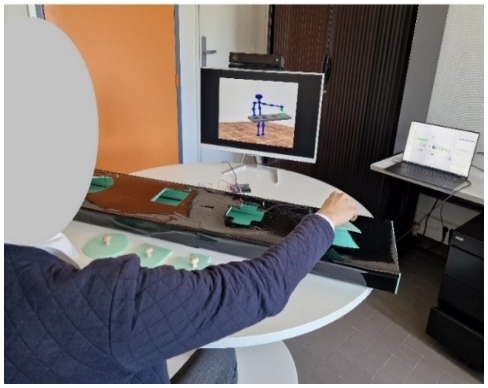

Fig. 14. The electronic and computer-based version of the Shape Sorter Drum Task (from [7])

devices and other types of input devices. In both cases, the fusion of events has to be described exhaustively and precisely. The ICO formal notation enables to smoothly integrate both types of multimodal interactions, and to verify their behavior against the requirements. This is why this step of the process also relies on the ICO formal notation. The outcome of the combination of the modalities in the input chain manager is the formal ICO description of the input chain manager as demonstrated in [33]. This is one of the key benefits of using a formal notation as ICO which is amenable to formal verification.

## 6 CASE STUDY: MULTI-DEVICES FOR POST-STROKE REHABILITATION

We applied the stepwise process for the design and development of interactive applications requiring specific input devices (including the generic process for integrating Frame-based input devices) to the integration of several devices for a post-stroke rehabilitation exercise setup. We reused the work from Carayon et al. [17] and adapted it to another specific rehabilitation goal. Carayon et al. [17] implemented a goal-based rehabilitation setup using two input devices: a gesture recognition device and an electronic sorter drum. This setup aims at training post-stroke patient to recognize shapes and to sort them while holding prescribed body postures. This case study presents how we first implemented the existing setup and then adapted it to reach additional rehabilitation goals and constraints.

### 6.1 Overview of the existing electronic version of the shape sorter drum task

The electronic version of the shape sorter drum [17] aims to guide the rehabilitation exercise as well as to monitor patient performance (shown in Figure 14). On a screen, the patient is asked to put the correct shape in the hole of the same shape. On insertion, a red LED is on if it is incorrect, a green one otherwise. The therapist, in real-time, can see on a dedicated screen what exercise the patient is doing and its current performance.

The electronic version of the shape sorter drum task is composed of:

- A shape sorter drum embedding a pair of red and green LEDs above each hole, as well as a sensor in each hole to detect what piece is falling in the hole.
- An Phidget board that connects the LEDs and the sensor from the shape sorter drum to a computer
- A Kinect device that senses the posture of the patient

### 6.2 Application of the stepwise process for design and development of interactive applications requiring specific input devices

Our main user requirement was to support the training and assessment of fine motor control in hand joint. Finger extension is one of the fine grain motor functions most likely to be impaired and is impeded by inappropriate flexor activation in upper extremity compensatory movements [26]. This means that one of the recurrent problems for post-stroke patients is having difficulties opening a hand.

*6.2.1 Prototyping.* We first implemented the shape sorter setup presented in [17] and performed preliminary user testing with healthy users. In addition, we performed a review of the setup with post-stroke rehabilitation experts. The outcome is a list of requirements for the informal interaction techniques, generic input devices and modalities:

- Req. 1: In addition to the tracking of the body posture, the setup shall track precisely each hand's movement (hand closed, hand open). This tracking has to be performed precisely to assess the patient's progress in fine grain motricity of finger joints.
- Req. 2: The setup shall be able to recognize, in a given interval of time (during a few seconds), that the patient holds a prescribed body posture, as well as that the patient opens the hand when dropping a shape above a hole, as well as the dropped shape.
- Req. 3: The setup shall differentiate the patient from another person passing near or behind the patient. The patient is exercising either in a public space (exercise room in a hospital) or at home, where other persons are likely to pass by.

About Req. 3, we actually faced the issue while testing the actual version of the shape sorter drum. If another person passes by or behind the user, the posture is wrongly detected as some parts of the person passing by are interpreted as new postures for a couple of joints. This compromises the guidance function in the application, as well as the information recorded for the analysis of the patient's performance.

*6.2.2 Identify input devices.* The existing setup supports the shape-sorting drum task, as well as the recognition of body postures. According to req. 1, the Kinect device, as well as the electronic shape sorter drum and the Phidget board are thus identified as concrete input devices to integrate. The Kinect device neither supports the precise tracking of a hand movement, nor the position of finger joints (req. 2). For that purpose, we need another device and the Leap is able to do this.

*6.2.3 Integrate Frame-based input devices.* The application of the generic process for integrating the Kinect v2 and Leap Frame-based devices is presented in sections 3 and 4.2. The outcome of the application of the process is the components models and generic transducers for both Frame-based devices. These component models have been presented earlier in the paper, in Figure 4 for the Kinect and in Figure 11 for the Leap.

*6.2.4 Produce specific transducers.* The production of the specific transducer starts by reducing the generic transducer to the exact need of the application. For the Kinect v2 input device, given the requirements, there is one event sink, "FRAME_PULLED", four event sources, "BODY_DETECTED", "BODY_UPDATED", "BODY_LOST" and "FRAME_PULLED". It is important to note that the event sink is also an event source to ensure that the specific transducer is Sly (as explained in section 3.4).

Figure 15 presents the ICO model describing the behavior of the specific transducer of the Kinect v2 for the target setup. The upper part of the model, surrounded with a red rectangle and labelled "Initialization" aims to instantiate the transducer and wait for its initialization to be complete. Then, it awaits the "FRAME_PULLED" event (listed in Figure 16 a)). The "FRAME_PULLED" event, includes a parameter which is an array of 6 potential bodies. If a body is tracked, the body will

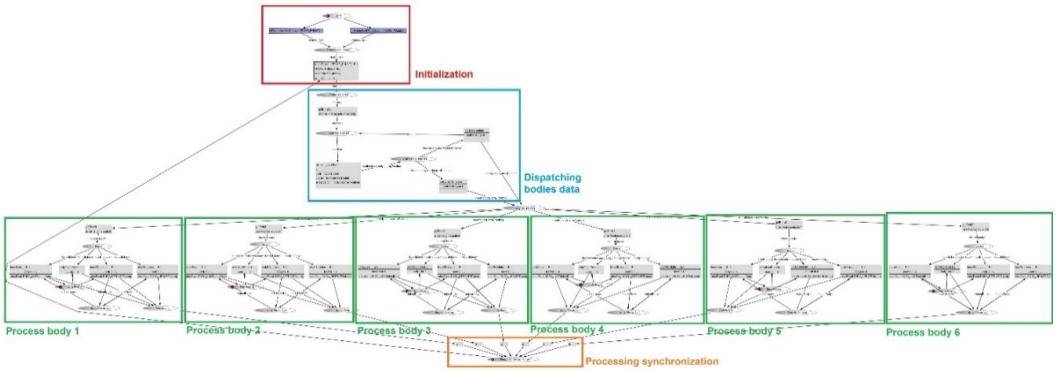


Fig. 15. Behavior (ICO model) of the specific transducer of the Kinect v2 meeting the application needs

contain a unique tracking ID and will contain tracked data. Otherwise, it will not contain any data and will have its tracking ID set to 0. When a body is detected, it is not possible to predict in which order a detected body will be inserted in the array; it can be at any position in the array. However, if a body is detected, it will stay in the same array position as long as it is detected, meaning that each position of the array must be computed separately. When receiving this "FRAME_PULLED" event, the part of the ICO model surrounded with a blue rectangle and labelled "Dispatching bodies data" extracts the six elements in the array and dispatches them in the six different green areas labelled "Process body 1 … 6".

Figure 16 presents an extract of the ICO model describing the behavior of the specific transducer of the Kinect v2 for the target setup. This extract describes the processing of the first body in the array. Initially the place "Body0Empty" contains an empty token, meaning that nothing is detected. If a body is detected, the "Body0Empty" token is removed and the token "body" goes through a transition raising a "BODY_DETECTED(body)" event before being stored in the "Body0Detected". When a new body is received, if the first body is still detected, it will remove the token "body" from the place and replace it with the token "latest_body", raising the event "BODY_UPDATED(latestBody)" event. On the other hand if the latest body retrieved is not tracked anymore, it will remove the token "body" from the "Body0Detected" and raise the event "BODY_LOST(body)", and store an empty token in the place "Body0Empty". When the processing is done, an empty token is stored in the "Body0Done". The same processing applies to the five other bodies. When all six bodies are done processing, the tokens in the places "Body1…6ProcessingDone" are passed in the part of the model surrounded with an orange rectangle and labelled "Processing synchronization" in Figure 17. The body processing is done and the transducer is ready to get the next frame, thus repeating the process.

Figure 17 presents the ICO model describing the behavior of the specific transducer of the Leap for the target setup. The upper part of the model, surrounded with a red rectangle and labelled "Init" handles the initialization of the generic transducer and the subscription to the "onFrame" event, which contains hands information. The Leap Motion provides an array data structure that contains between zero and two elements and also the size of the array. The Leap Motion can detect maximum one right hand and one left hand but it is not possible to know which hand will be in which position of the array. The part of the ICO model surrounded with a blue rectangle and labelled "Checking number of hands" checks how many hands are available in the array. Depending on how much hands are detected, there are three possible outcomes. First possible outcome is to raise the event "frameEmptyReceived(frame)" and continue to the section of the model surrounded

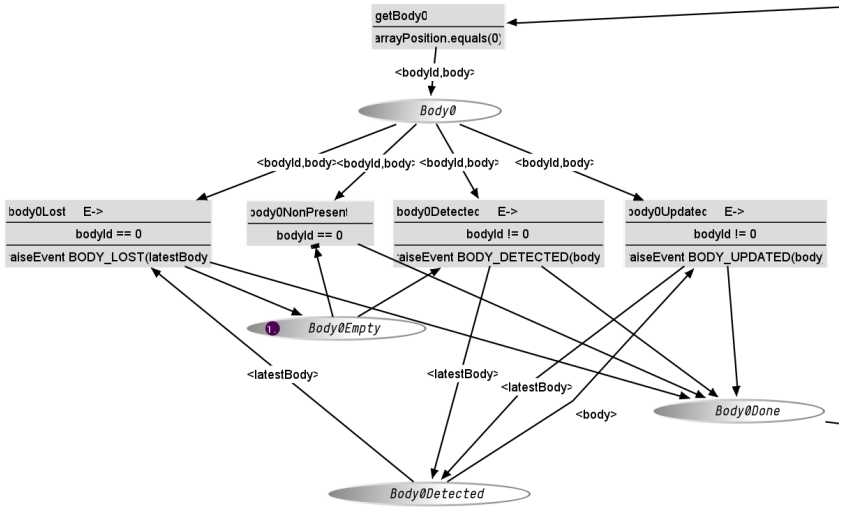


Fig. 16. Behavior (ICO model) of the processing of one of the six potential bodies detected by the Kinect v2

with an orange rectangle and labelled “No hands”. Second possible outcome is to raise the event “frameOneHandReceived(frame)” and to continue to the section “One hand”. Third possible outcome is to raise the event “frameTwoHandsReceived(frame)” and continue to the section “Two hands”. In the parts of the ICO model labelled “One hand” and “Two hands”, the state of each hand (left or right) in the array is checked and sent to either the part of the ICO model surrounded with a green rectangle and labelled “Processing left hand” or to the part of the ICO model surrounded with a green rectangle and labelled “Processing right hand”.

Figure 18 shows in details the part of the ICO model named “Processing left hand” in Figure 17. This part of the ICO model is quite similar to the way the specific transducer of the Kinect v2 handles the body detection. Initially a token is in the place “HandLeftNotDetected”, and upon the detection of the left hand (arrival of a token in the place “HandLeft”), a token is put in the place “HandLeftDetected” after the event “leftHandDetected(leapHand)” has been raised. The token in the place “HandLeftNotDetected” and the token in the place “HandLeft” are removed. If in the next frame a left hand is still available, with the arrival of a new token in the place “HandLeft”, a new token replaces the previous one in the place “HandLeftDetected” after an event “leftHandUpdated(leapHand)” has been raised. The token in the place “HandLeft” is removed. If on the next frame no left hand was detected (inhibitor arc between the place “HandLeft” and the transition “handLeftLost_”) but a right hand was detected (a token is available in the place “HandRight”), a token is put in the place “LeftHandProcessingDone” after an event “leftHandLost(leapHand)” has been raised. If no hands at all were detected (a token is available in the place “NoHands”), a token is put in the place “LeftHandProcessingDone” after an event “handLeftLostNoHands” has been raised.

The specific transducer of the Leap applies the same type of processing to the right hand (part labeled “Processing right hand” in Figure 17) and when everything is processed, the transducer is ready to fetch the next frame.

According to req. 1, the setup has to detect the status of each hand (opened or closed). We thus completed the specific transducer of the Leap to match this requirement. Figure 19 presents an extract of the ICO model describing the behavior of the specific transducer of the Leap for the target setup. If the event “rightHandDetected” is received, the frame is considered valid and awaits the

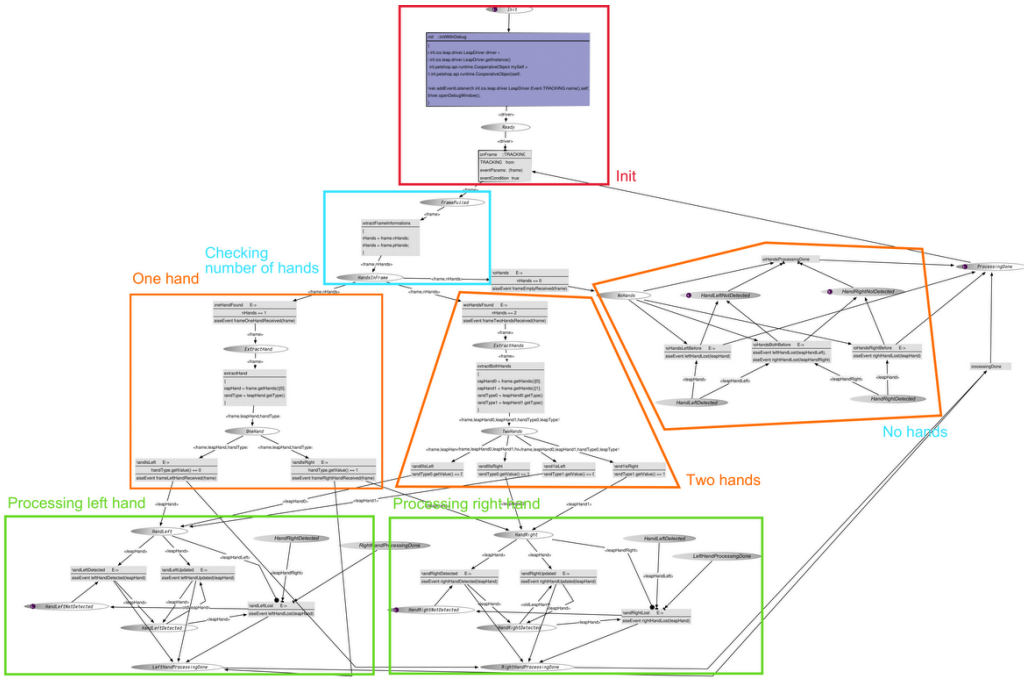


Fig. 17. Behavior (ICO model) of the specific transducer of the Leap for the target setup

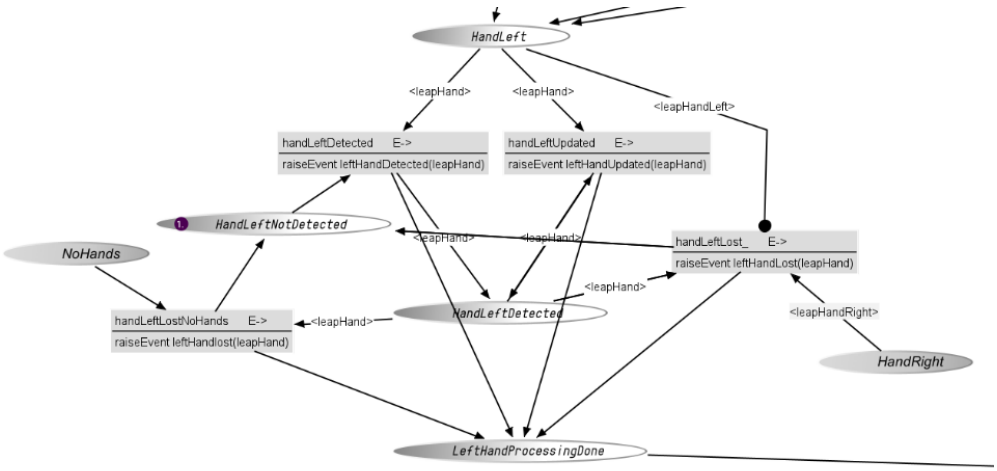


Fig. 18. Behavior (ICO model) of the processing of the left hand detected by the Leap

"leftHandDetected(leapHand)" and "rightHandDectected(leapHand)" in the "Left hand processing and the "Right hand processing" section. Initially, a hand is considered closed. When the hand state goes from "closed" to "opened" the transducer raise the event "leftHandOpened()" and when the hand state goes from "opened" to "closed" the transducer raise the event "rightHandClosed()".

When both hands events have been processed, the transducer is ready to receive the next frame.

According to req. 3, the setup shall differentiate the interacting body from another body passing near or behind the patient. The "main user" is considered in our case to be the first person detected by the Kinect and anyone else would be ignored. We thus completed the specific transducer of the Kinect to match this requirement.

Figure 20 presents the ICO model describing the behavior of the specific transducer of the Kinect for getting the first user being detected. Initially, no user is detected. The first time an event "BODY_DETECTED(body,bodyid)" is received (transition "bodyReceived: BODY_DETECTED"), an event "mainUserDetected(body)" is sent (transition "detectMainUser") and a token containing the body and its id is stored in the "UserBodyTracked" place. Upon receiving a BODY_UPDATED(body, bodyId) event (transition "bodyUpdated: BODY_UPDATED"), if the bodyId matches the currently tracked user, the event "mainUserUpdated(updatedBody)" is sent (transition "userUpdate") and the token containing the user is updated. Upon receiving a BODY_LOST(body,bodyId) event (transition "bodyLost: BODY_LOST"), if the bodyId matches the currently tracked user, the event "mainUserLost(body)" is sent (transition "userLost"), the token containing the body is discarded and an empty token is put in the place "NoUserDetected".

While the user is detected, any event "BODY_UPDATED(body,bodyId)" or "BODY_LOST(body,bodyId)" that does not match the user id is discarded because it corresponds to another person interfering and can be ignored (in compliance with req. 3).

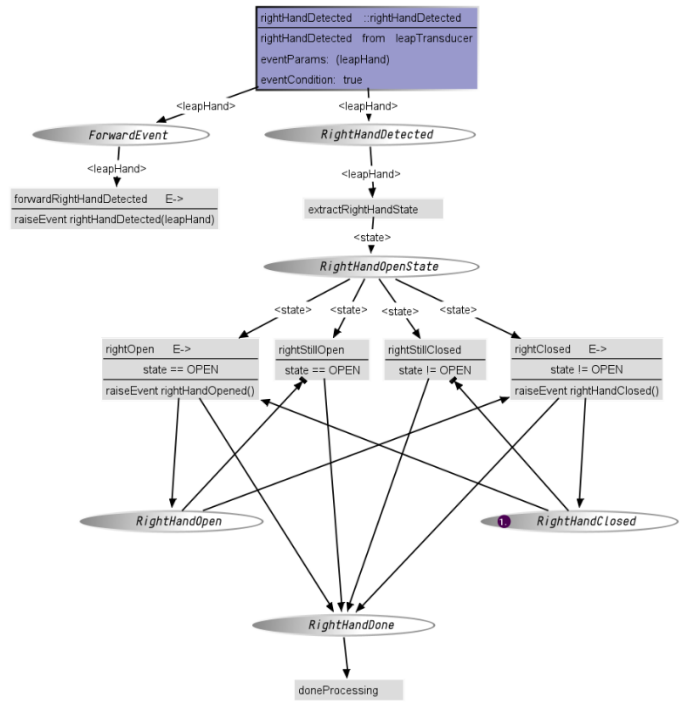


Fig. 19. Extract of the behavior (ICO model) of the specific transducer of the Leap for right hand open/close detection

*6.2.5 Combine modalities in the input chain manager.* In order to match requirement 2, the setup shall be able to recognize, in a given interval of time (during a few seconds), that the patient holds

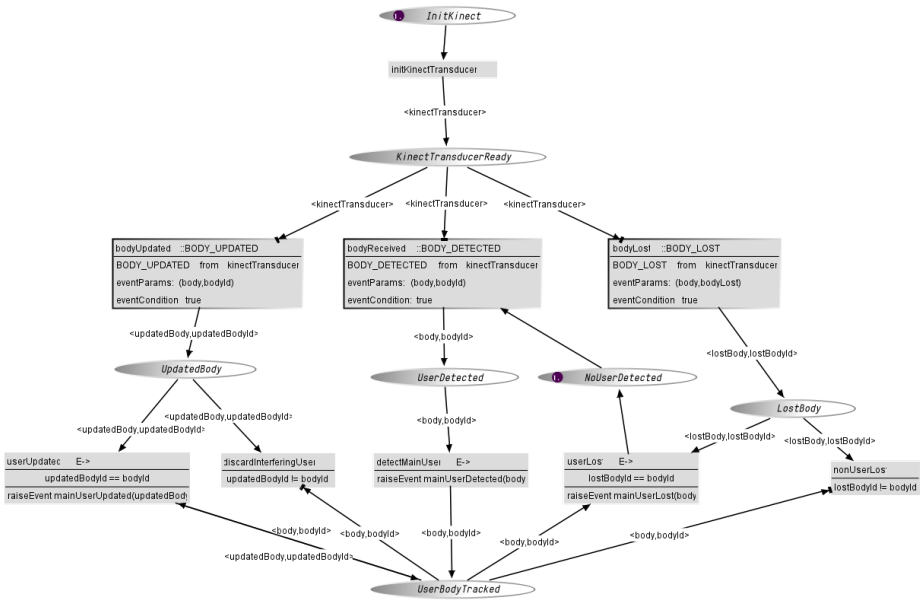


Fig. 20. Extract of the behavior (ICO model) of the specific transducer of the Kinect v2 for the detection of one person (and exclusion of other persons)

a prescribed body posture, as well as that the patient opens the hand when dropping a shape above a hole, as well as the dropped shape. This means that we need to combine modalities in an Input Chain Manager transducer of MIODMIT architecture (see Figure 1).

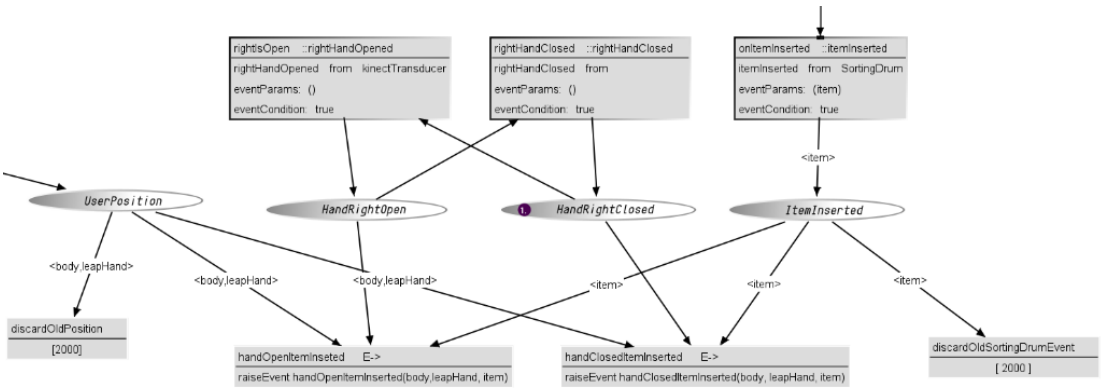


Fig. 21. Extract of the behavior (ICO model) of the input chain manager- full model here https://sites.google.com/view/eics2024beyondprototyping/

Figure 21 presents the extract of the ICO model describing the behavior of the input chain manager and in particular illustrates how it matches req. 2. If a user position (body posture and at least the right hand) is detected, when a token is in the place "UserPosition", and that a token is in the place "HandRightClosed" and that a token is in the place "ItemInserted", the transition "handCloseItemInserted" fires and raise the event "handClosedItemInserted" (with parameters "body",

"leapHand" and "item"). If a user position (body posture and at least the right hand) is detected, when a token is in the place "UserPosition", and that a token is in the place "HandRightOpen" and that a token is in the place "ItemInserted", the transition "handOpenItemInserted" fires and raise the event "handOpenItemInserted" (with parameters "body", "leapHand" and "item"). A token in the place "UserPosition" is consumed or removed after 2 seconds (transition "discardOldPosition" connected from the place "UserPosition"). In the same way, a token in the place "ItemInserted" is consumed or removed after 2 seconds (transition "discardOldSortingDrumEvent" connected from the place "ItemInserted"). Both transitions enable to ensure that req. 2 is met, by enabling the firing of the events "handClosedItemInserted" and "handOpenItemInserted" only if the patient holds a prescribed body posture, as well as that the patient opens the hand when dropping a shape above a hole, as well as the dropped shape, all of these within 2 seconds. The events "handClosedItemInserted" and "handOpenItemInserted" are sinks for the application managing the full setup of the Shape Sorter Drum Task. In this model we can that that it is possible to receive an event "handClosed" and another event "ItemInserted" in the same temporal window. This can be seen as contradictory with respect to RQ2. In fact, RQ2, is checked at the application level. Receiving these two events together is feasible with the input devices but results in a failure by the user to perform the exercise as expected.

*6.2.6 Benefits of the approach.* The case study has demonstrated that the generic process can be applied to go from the requirements of the application to the production of an entire chain of transducers processing information sensed by a Frame-based input device. We have also demonstrated that these chain flows can be fused in a dedicated multimodal transducer (the Input Chain Manager).

The case study has also demonstrated the importance of the behavioral side of the transducers which is usually not presented nor discussed, even in the documentation of the input devices. We have used the ICO formalism for describing this behavior as those models can be verified and executed. The running of the models and the application will be demonstrated at the conference.

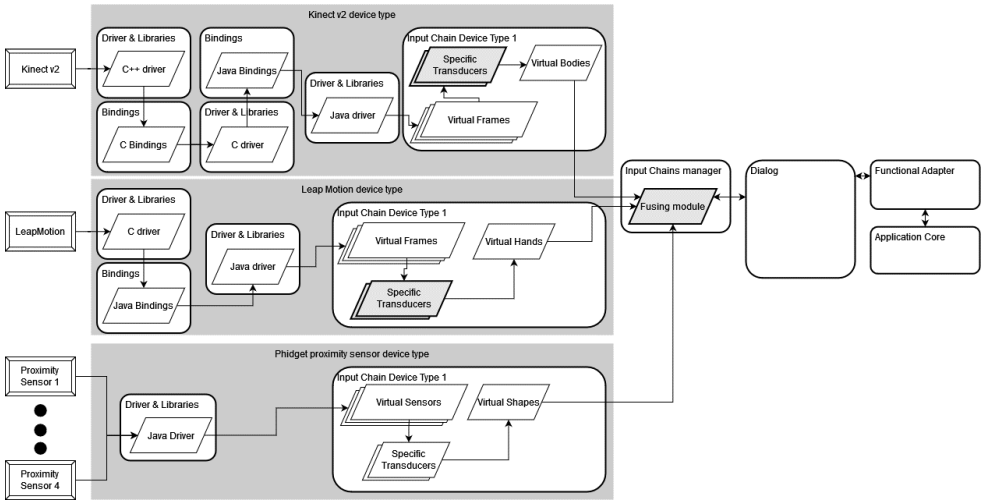


Fig. 22. MIODMIT architecture of the case study (grey oblique parallelograms in the model presented above are presented complete and in high-resolution in https://sites.google.com/view/eics2024beyondprototyping/)

## 7 LESSONS LEARNT

As shown in section 2, multiple frameworks and toolkits have been proposed for exploiting Frame-based devices. In all of them, their authors acknowledge the difficulty of engineering applications that uses such Frame-based devices and this is the main motivation for the development of their work.

Table 1 presents, in a tabular form, a comparison of the benefits and limitation of these approaches and how the current paper relates to them. While they all provide clear contributions, they tend to focus either on specific interaction techniques or specific devices. In addition, they are also bound to specific languages and provide limited extensibility in term of services, events and interaction techniques. Beyond, while some of them (see column 9) define and implement some custom interactions that are not directly offered with the device, very few of them (see column 10) offer the possibility define additional custom interactions.

Even though improving the engineering is the main reason for their work, very few have addressed the verification of the processing of the events flow and the interaction techniques on top of it. This is a clear limitation as the input chain processing is critical as it is exploited each time an interaction with the user takes place. If one element of the input chain is failing, it means that user input will be ignored or processed inadequately. While it can be understandable that formal approaches could be challenging in terms of complexity, the testing of the input chain should have been addressed, as for instance in [18] for "standard" input devices.

While our approach requires some work if one starts from scratch, its main interest comes from the modularity provided by the extended MIODMIT architecture and limited work is required for modifying an existing interactive application architecture with MIODMIT. For example, if we want to change our target language, we just need to re-write code in the block "Bindings".

Finally, related work on the use of multiple Frame-based devices in section 2.2 has shown that being able to exploit this type of devices (in an independent or in a multimodal manner) is very promising and support user interactions that are otherwise impossible. However, this related work focuses on the construction of a solution to a specific problem and do not propose systematic ways to integrate them. This means that the developer willing to use them must develop everything from the identification of the capabilities of the device (via the analysis of the associated API) to the implementation of the designed interactions.

By using the architecture and the process presented in the paper, developers will be able to locate place of work are able identify where the difficulties lay, and are supported by a systematic stepwise process. We believe that such support will favor the integration of Frame-based devices and the deployment of more sophisticated interaction techniques as the approach supports reuse of devices and their input chain (though the CCM description) and tuning of interaction (through small modification of the ICOs models). Frame-based. It is clear however that the full exploitation of the approach requires knowledge in CCM and the capability of reading Petri nets models. Other notations such as state machines could have been used but to the detriment of being able to describe concurrent behaviors which is required for multimodal interactions as demonstrated in [47].

One of the goals of using a formal Petri nets-based notation (ICOs) for the description of the behavioral models of the components of MIODMIT architecture, is that they are amenable to formal verification. Formal verification supports the detection of defects in the models as well as model-based performance evaluation [32] of communicating components. How to perform formal verification on ICO models has been presented in [39]. It is also possible to check safety and liveness properties on a single model (see [43]) as well as on a set of cooperating models (ensuring for instance that demand of a given model matches the offer of the other models [8]).

Table 1. Summary of the specifications and benefits provided by related work compared to our approach

| | Supported Devices | Supported Languages | Multi-Device Support | Possibility to add other languages | Possibility to add other devices | Possibility to use multiple devices | Provides custom data and events | Provide custom interactions | Possibility to define additional interactions | Verification |
|---|---|---|---|---|---|---|---|---|---|---|
| SoD-toolkit [50] | Kinect v1 Kinect v2 Apple iBeacon LeapMotion | Javascript | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Quantum Leap [51] | Leap Motion | Javascript | ✗ | ✗ | ✓ | ~ | ✓ | ✓ | ✓ | ✗ |
| XDKinect [40] | Kinect v2 | Javascript | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| Xkin [44] | Kinect v2 | Javascript | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ |
| KinZ [52] | Kinect Azure | Matlab Python | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Current paper | Any in theory (Kinect V2 and Leap in the paper) | Any in theory (Java in the paper) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

## 8 CONCLUSION

This paper builds on top of recent contributions in the area of interactive systems engineering. It exploits and extend MIODMIT architecture [19] to encompass the specificities of Frame-based input devices. We have shown that these input devices might be demanding in terms of computing resources especially via the sensing of up to sixty frames per seconds per embedded video camera. In order to exploit those devices, we have proposed a process for producing generic models of their driver and also for producing specific models of these drivers (in order, for instance, to limit the sensing to the needs of the interactive application exploiting them). The generic models are presented using the ICO formalism which can be translated to code through compiling (as presented in [48] – but this is not addressed in the current paper) or through interpretation using PetShop tool [16]. Such abstract models and the associated definition process aims at supporting researchers and engineers to embed such devices in their prototypes and systems as all the states and state changes are made explicit.

We have also proposed a generic way to embed those devices in a manner that is independent from the programing language used to develop the interactive application by a systematic definition of binders from the language of the API of the devices to the language of the interactive application.

These contributions go beyond current state of the art in Frame-based interactive applications as the process and the associated models support complete and unambiguous description of the entire input chain (including device drivers, transducers and interaction techniques) and making the models amenable to formal verification supporting dependability to a place where the contributions are through prototypes and demonstrators exploiting knowledge which is mainly craft-based.

We have demonstrated the application of the process, the architecture and the ICO formalism on a case study from the healthcare domain where dependability is at least as important as usability. This post-stroke rehabilitation application exploits two Frame-based input devices (a Kinect and a Leap Motion) as well as a dedicated one for physical interactions. These devices are used in a multimodal synergistic way to track accurately patients' exercises. As the goal of the paper is focusing on the engineering aspects, clinical aspects have not been presented in the paper.

Compared to other solutions dealing with the integration of Frame-based devices (such as QuantumLeap[51]), the solution presented in the current paper is generic for any kind of device, language and interaction technique. On the application side, the current paper goes beyond the contribution of Carayon et al. [7] in terms of the accuracy of patients tracking (both detecting perturbations in the environment when exercises are performed and track hand movement together with upper limb ones).

Due to space constraints the paper was focused on the input chain and left aside the output one. That chain is also critical in order to present information to the patient. As presented in MIODMIT the output chain should handle immediate feedback (how the actions performed on the input devices are presented to the user e.g. the mouse cursor) but also the semantic feedback coming from the interactive application itself (e.g. the exercise has been adequately performed or need to be redone due to perturbations). Carayon et al. [7] presented an informal approach to this critical problem of rehabilitation which should be formalized and better supported by the tools.

## 9 ACKNOWLEDGMENT

The authors would like to sincerely thank the anonymous reviewers for their work on the submitted version of the paper. This work was partially funded by the ANR project ANR-21-CE33-0008 and the CNES 21055/00.

## REFERENCES

[1] 2014. IBodyFrameSource Interface. https://learn.microsoft.com/fr-fr/previous-versions/windows/kinect/dn772943(v=ieb.10)

[2] 2014. Kinect API Overview. https://learn.microsoft.com/fr-fr/previous-versions/windows/kinect/dn782033(v=ieb.10)

[3] 2023. Discontinuing Kinect and Azure Kinect. https://techcrunch.com/2023/08/22/microsoft-cant-stop-discontinuing-kinect/

[4] 2024. Leap Motion Controller - UltraLeap. https://leap2.ultraleap.com/leap-motion-controller/

[5] Johnny Accot, Stéphane Chatty, Sébastien Maury, and Philippe Palanque. 1997. Formal Transducers: Models of Devices and Building Bricks for the Design of Highly Interactive Systems. In *Design, Specification and Verification of Interactive Systems '97*, W. Hansmann, W. T. Hewitt, W. Purgathofer, Michael Douglas Harrison, and Juan Carlos Torres (Eds.). Springer Vienna, Vienna, 143–159. https://doi.org/10.1007/978-3-7091-6878-3_10

[6] Johnny Accot, Stéphane Chatty, and Philippe Palanque. 1996. A Formal Description of Low Level Interaction and its Application to Multimodal Interactive Systems. In *Design, Specification and Verification of Interactive Systems '96*, W. Hansmann, W. T. Hewitt, W. Purgathofer, Francois Bodart, and Jean Vanderdonckt (Eds.). Springer Vienna, Vienna, 92–104. https://doi.org/10.1007/978-3-7091-7491-3_5

[7] Carayon Axel, Juan E. Garrido, Célia Martinie, Philippe Palanque, Eric Barboni, María Dolores Lozano, and Víctor M. R. Penichet. 2023. Engineering Rehabilitation: Blending Two Tool-supported Approaches to Close the Loop from Tasks-based Rehabilitation to Exercises and Back Again. *Proceedings of the ACM on Human-Computer Interaction* 7, EICS (June 2023), 1–23. https://doi.org/10.1145/3593229

[8] Rémi Bastide, Philippe Palanque, Ousmane Sy, and David Navarre. 2000. Formal specification of CORBA services: experience and lessons learned. *ACM SIGPLAN Notices* 35, 10 (Oct. 2000), 105–117. https://doi.org/10.1145/354222.353179

[9] Rémi Bastide, Ousmane Sy, David Navarre, and Philippe Palanque. 2000. A Formal Specification of the CORBA Event Service. In *Formal Methods for Open Object-Based Distributed Systems IV*, Scott F. Smith and Carolyn L. Talcott (Eds.). Vol. 49. Springer US, Boston, MA, 371–395. https://doi.org/10.1007/978-0-387-35520-7_19

[10] Rémi Bastide, Ousmane Sy, and Philippe Palanque. 2000. A formal notation and tool for the engineering of CORBA systems. *Concurrency: Practice and Experience* 12, 14 (Dec. 2000), 1379–1403. https://doi.org/10.1002/1096-9128(20001210)12:14<1379::AID-CPE514>3.0.CO;2-B

[11] Rémi Bastide, Ousmane Sy, and Philippe Palanque. 2000. A formal notation and tool for the engineering of CORBA systems. *Concurrency: Practice and Experience* 12, 14 (Dec. 2000), 1379–1403. https://doi.org/10.1002/1096-9128(20001210)12:14<1379::AID-CPE514>3.0.CO;2-B

[12] Renaud Blanch and Michel Beaudouin-Lafon. 2006. Programming rich interactions using the hierarchical state machine toolkit. In *Proceedings of the working conference on Advanced visual interfaces - AVI '06*. ACM Press, Venezia, Italy, 51. https://doi.org/10.1145/1133265.1133275

[13] Richard A. Bolt. 1980. “Put-that-there”: Voice and gesture at the graphics interface. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’80*. ACM Press, Seattle, Washington, United States, 262–270. https://doi.org/10.1145/800250.807503

[14] W. Buxton and B. Myers. 1986. A study in two-handed input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Boston Massachusetts USA, 321–326. https://doi.org/10.1145/22627.22390

[15] Ziyun Cai, Jungong Han, Li Liu, and Ling Shao. 2017. RGB-D datasets using microsoft kinect or similar sensors: a survey. *Multimedia Tools and Applications* 76 (2017), 4313–4355.

[16] José Creissac Campos, Camille Fayollas, Michael D. Harrison, Célia Martinie, Paolo Masci, and Philippe Palanque. 2020. Supporting the Analysis of Safety Critical User Interfaces: An Exploration of Three Formal Tools. *ACM Transactions on Computer-Human Interaction* 27, 5 (Oct. 2020), 1–48. https://doi.org/10.1145/3404199

[17] Alexandre Canny, Elodie Bouzekri, Célia Martinie, and Philippe Palanque. 2019. Rationalizing the Need of Architecture-Driven Testing of Interactive Systems. In *Human-Centered Software Engineering*, Cristian Bogdan, Kati Kuusinen, Marta Kristín Lárusdóttir, Philippe Palanque, and Marco Winckler (Eds.). Vol. 11262. Springer International Publishing, Cham, 164–186. https://doi.org/10.1007/978-3-030-05909-5_10

[18] Alexandre Canny, Célia Martinie, David Navarre, Philippe Palanque, Eric Barboni, and Christine Gris. 2021. Engineering Model-Based Software Testing of WIMP Interactive Applications: A Process based on Formal Models and the SQUAMATA Tool. *Proceedings of the ACM on Human-Computer Interaction* 5, EICS (May 2021), 1–30. https://doi.org/10.1145/3461729

[19] Martin Cronel, Bruno Dumas, Philippe Palanque, and Alexandre Canny. 2019. MIODMIT: A Generic Architecture for Dynamic Multimodal Interactive Systems. In *Human-Centered Software Engineering*, Cristian Bogdan, Kati Kuusinen, Marta Kristín Lárusdóttir, Philippe Palanque, and Marco Winckler (Eds.). Vol. 11262. Springer International Publishing, Cham, 109–129. https://doi.org/10.1007/978-3-030-05909-5_7

[20] Florian Echtler and Gudrun Klinker. 2008. A multitouch software architecture. In *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges*. ACM, Lund Sweden, 463–466. https://doi.org/10.1145/1463160.1463220

[21] Matthias Grimmer, Manuel Rigger, Lukas Stadler, Roland Schatz, and Hanspeter Mössenböck. 2013. An efficient native function interface for Java. In *Proceedings of the 2013 International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools*. ACM, Stuttgart Germany, 35–44. https://doi.org/10.1145/2500828.2500832

[22] Tovi Grossman and Ravin Balakrishnan. 2005. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor’s activation area. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Portland Oregon USA, 281–290. https://doi.org/10.1145/1054972.1055012

[23] Arnaud Hamon, Phil Palanque, Raphaël André, Eric Barboni, Martin Cronel, and David Navarre. 2014. Multi-Touch interactions for control and display in interactive cockpits: issues and a proposal. In *Proceedings of the International Conference on Human-Computer Interaction in Aerospace*. ACM, Santa Clara California, 1–10. https://doi.org/10.1145/2669592.2669650

[24] Arnaud Hamon, Philippe Palanque, Martin Cronel, Raphaël André, Eric Barboni, and David Navarre. 2014. Formal modelling of dynamic instantiation of input devices and interaction techniques: application to multi-touch interactions. In *Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, Rome Italy, 173–178. https://doi.org/10.1145/2607023.2610286

[25] Lode Hoste, Bruno Dumas, and Beat Signer. 2011. Mudra: a unified multimodal interaction framework. In *Proceedings of the 13th international conference on multimodal interfaces*. ACM, Alicante Spain, 97–104. https://doi.org/10.1145/2070481.2070500

[26] D.G. Kamper, R.L. Harvey, S. Suresh, and W.Z. Rymer. 2003. Relative contributions of neural mechanisms versus muscle mechanics in promoting finger extension deficits following stroke. *Muscle & Nerve* 28, 3 (Sept. 2003), 309–318. https://doi.org/10.1002/mus.10443

[27] Spyros Kousidis, Casey Kennington, Timo Baumann, Hendrik Buschmeier, Stefan Kopp, and David Schlangen. 2014. A Multimodal In-Car Dialogue System That Tracks The Driver’s Attention. In *Proceedings of the 16th International Conference on Multimodal Interaction*. ACM, Istanbul Turkey, 26–33. https://doi.org/10.1145/2663204.2663244

[28] Radoslava Kraleva and Velin Kralev. 2016. On model architecture for a children’s speech recognition interactive dialog system. (2016). https://doi.org/10.48550/ARXIV.1605.07733

[29] Pradeep Kumar, Himaanshu Gauba, Partha Pratim Roy, and Debi Prosad Dogra. 2017. A multimodal framework for sensor based sign language recognition. *Neurocomputing* 259 (Oct. 2017), 21–38. https://doi.org/10.1016/j.neucom.2016.08.132

[30] Pradeep Kumar, Himaanshu Gauba, Partha Pratim Roy, and Debi Prosad Dogra. 2017. Coupled HMM-based multi-sensor data fusion for sign language recognition. *Pattern Recognition Letters* 86 (Jan. 2017), 1–8. https://doi.org/10.1016/j.patrec.2016.12.004

[31] Bass L, Little R, Pellegrino R, Reed S, Seacord R, Sheppard S, and Szezur M. R. 1991. The arch model : Seeheim revisited.
[32] Xavier Lacaze, Philippe Palanque, David Navarre, and Rémi Bastide. 2002. Performance Evaluation as a Tool for Quantitative Assessment of Complexity of Interactive Systems. In *Interactive Systems:Design, Specification, and Verification*, Gerhard Goos, Juris Hartmanis, Jan Van Leeuwen, Peter Forbrig, Quentin Limbourg, Jean Vanderdonckt, and Bodo Urban (Eds.). Vol. 2545. Springer Berlin Heidelberg, Berlin, Heidelberg, 208–222. https://doi.org/10.1007/3-540-36235-5_16
[33] Jean-François Ladry, David Navarre, and Philippe Palanque. 2009. Formal description techniques to support the design, construction and evaluation of fusion engines for sure (safe, usable, reliable and evolvable) multimodal interfaces. In *Proceedings of the 2009 international conference on Multimodal interfaces*. ACM, Cambridge Massachusetts USA, 185–192. https://doi.org/10.1145/1647314.1647347
[34] Bo Li, Chao Zhang, Cheng Han, and Baoxing Bai. 2018. Fingertip data fusion of Kinect v2 and leap motion in unity. *Ingénierie des systèmes d'information* 23, 6 (Dec. 2018), 143–159. https://doi.org/10.3166/isi.23.6.143-159
[35] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. 2014. Hand gesture recognition with leap motion and kinect devices. In *2014 IEEE International Conference on Image Processing (ICIP)*. IEEE, Paris, France, 1565–1569. https://doi.org/10.1109/ICIP.2014.7025313
[36] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. 2016. Hand gesture recognition with jointly calibrated Leap Motion and depth sensor. *Multimedia Tools and Applications* 75, 22 (Nov. 2016), 14991–15015. https://doi.org/10.1007/s11042-015-2451-6
[37] Daniel Martin, Sandra Malpica, Diego Gutierrez, Belen Masia, and Ana Serrano. 2022. Multimodality in VR: A Survey. *Comput. Surveys* 54, 10s (Jan. 2022), 1–36. https://doi.org/10.1145/3508361
[38] Brad A. Myers, Ashley Lai, Tam Minh Le, YoungSeok Yoon, Andrew Faulring, and Joel Brandt. 2015. Selective Undo Support for Painting Applications. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, Seoul Republic of Korea, 4227–4236. https://doi.org/10.1145/2702123.2702543
[39] David Navarre, Philippe Palanque, Jean-Francois Ladry, and Eric Barboni. 2009. ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. *ACM Transactions on Computer-Human Interaction* 16, 4 (Nov. 2009), 1–56. https://doi.org/10.1145/1614390.1614393
[40] Michael Nebeling, Elena Teunissen, Maria Husmann, and Moira C. Norrie. 2014. XDKinect: development framework for cross-device interaction using kinect. In *Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, Rome Italy, 65–74. https://doi.org/10.1145/2607023.2607024
[41] Albert Ng, Julian Lepinski, Daniel Wigdor, Steven Sanders, and Paul Dietz. 2012. Designing for low-latency direct-touch input. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, Cambridge Massachusetts USA, 453–464. https://doi.org/10.1145/2380116.2380174
[42] Object Management Group. 1995. The Common Object Request Broker (CORBA): Architecture and Specification. https://www.omg.org/spec/CORBA/2.5/PDF
[43] Ph Palanque and R. Bastide. 1995. Verification of an Interactive Software by Analysis of its Formal Specification. In *Human—Computer Interaction*, Knut Nordby, Per Helmersen, David J. Gilmore, and Svein A. Arnesen (Eds.). Springer US, Boston, MA, 191–196. https://doi.org/10.1007/978-1-5041-2896-4_32
[44] Fabrizio Pedersoli, Sergio Benini, Nicola Adami, and Riccardo Leonardi. 2014. XKin: an open source framework for hand pose and gesture recognition using kinect. *The Visual Computer* 30, 10 (Oct. 2014), 1107–1122. https://doi.org/10.1007/s00371-014-0921-x
[45] Benoît Penelle and Olivier Debeir. 2014. Multi-sensor data fusion for hand tracking using Kinect and Leap Motion. In *Proceedings of the 2014 Virtual Reality International Conference*. ACM, Laval France, 1–7. https://doi.org/10.1145/2617841.2620710
[46] Günther E. Pfaff (Ed.). 1985. *User Interface Management Systems: Proceedings of the Workshop on User Interface Management Systems held in Seeheim, FRG, November 1–3, 1983*. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-70041-5
[47] Palanque Philippe and Schyn Amélie. 2003. A Model-Based Approach for Engineering Multimodal Interactive. In *IFIP TC13 International Conference on Human-Computer Interaction*. Zurich.
[48] Bastide Rémi and Palanque Philippe. 1996. Implementation Techniques for Petri Net Based Specifications of Human-Computer Dialogues.
[49] Kizito Salako and Lorenzo Strigini. 2014. When Does "Diversity"' in Development Reduce Common Failures? Insights from Probabilistic Modeling. *IEEE Transactions on Dependable and Secure Computing* 11, 2 (March 2014), 193–206. https://doi.org/10.1109/TDSC.2013.32
[50] Teddy Seyed, Alaa Azazi, Edwin Chan, Yuxi Wang, and Frank Maurer. 2015. SoD-Toolkit: A Toolkit for Interactively Prototyping and Developing Multi-Sensor, Multi-Device Environments. In *Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces - ITS '15*. ACM Press, Madeira, Portugal, 171–180. https://doi.org/10.1145/2817721.2817750

[51] Arthur Sluÿters, Mehdi Ousmer, Paolo Roselli, and Jean Vanderdonckt. 2022. QuantumLeap, a Framework for Engineering Gestural User Interfaces based on the Leap Motion Controller. *Proceedings of the ACM on Human-Computer Interaction* 6, EICS (June 2022), 1–47. https://doi.org/10.1145/3532211

[52] Juan R. Terven and Diana M. Córdova-Esparza. 2021. KinZ an Azure Kinect toolkit for Python and Matlab. *Science of Computer Programming* 211 (Nov. 2021), 102702. https://doi.org/10.1016/j.scico.2021.102702

[53] Ramadevi Vennelakanti, Prasenjit Dey, Ankit Shekhawat, and Phanindra Pisupati. 2011. The picture says it all!: multimodal interactions and interaction metadata. In *Proceedings of the 13th international conference on multimodal interfaces*. ACM, Alicante Spain, 89–96. https://doi.org/10.1145/2070481.2070499

[54] Norman Villaroman, Dale Rowe, and Bret Swan. 2011. Teaching natural user interaction using OpenNI and the Microsoft Kinect sensor. In *Proceedings of the 2011 conference on Information technology education*. ACM, West Point New York USA, 227–232. https://doi.org/10.1145/2047594.2047654

[55] Oliver Wasenmüller and Didier Stricker. 2017. Comparison of Kinect V1 and V2 Depth Images in Terms of Accuracy and Precision. In *Computer Vision – ACCV 2016 Workshops*, Chu-Song Chen, Jiwen Lu, and Kai-Kuang Ma (Eds.). Vol. 10117. Springer International Publishing, Cham, 34–45. https://doi.org/10.1007/978-3-319-54427-4_3

[56] Yuanjie Wu, Lei Gao, Simon Hoermann, and Robert W. Lindeman. 2018. Towards Robust 3D Skeleton Tracking Using Data Fusion from Multiple Depth Sensors. In *2018 10th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games)*. IEEE, Wurzburg, 1–4. https://doi.org/10.1109/VS-Games.2018.8493443

[57] Yuanjie Wu, Yu Wang, Sungchul Jung, Simon Hoermann, and Robert W. Lindeman. 2019. Exploring the Use of a Robust Depth-sensor-based Avatar Control System and its Effects on Communication Behaviors. In *25th ACM Symposium on Virtual Reality Software and Technology*. ACM, Parramatta NSW Australia, 1–9. https://doi.org/10.1145/3359996.3364267

[58] Yuanjie Wu, Yu Wang, Sungchul Jung, Simon Hoermann, and Robert W. Lindeman. 2019. Towards an articulated avatar in VR: Improving body and hand tracking using only depth cameras. *Entertainment Computing* 31 (Aug. 2019), 100303. https://doi.org/10.1016/j.entcom.2019.100303

First Submitted February 2024; Revised April 2024; Accepted April 2024