Semantics-based Software Techniques for Maintainable Multimodal Input Processing in Real-time Interactive Systems

Martin Fischbach* University of Würzburg
Dennis Wiebusch† University of Würzburg
Marc Erich Latoschik‡ University of Würzburg

Abstract

Maintainability, i.e. reusability, modifiability, and modularity, is a critical non-functional quality requirement, especially for software frameworks. Its fulfilment is already challenging for low-interactive application areas. It is additionally complicated by complex system designs of Real-time Interactive Systems (RISs), required for Augmented, Mixed, and Virtual Reality, as well as computer games. If such systems incorporate AI methods, as required for the implementation of multimodal interfaces or smart environments, it is even further exacerbated. Existing approaches strive to establish software technical solutions to support the close temporal and semantic coupling required for multimodal processing and at the same time preserve a general decoupling principle between involved software modules. We present two key solutions that target the semantic coupling issue: (1) a semantics-based access scheme to principal elements of the application state and (2) the specification of effects by means of semantic function descriptions for multimodal processing. Both concepts are modeled in an OWL ontology. The applicability of our concepts is showcased by a prototypical implementation and explained by an interaction example that is applied for two application areas.

Index Terms: D.2.11 [Software Engineering]: Software Architectures—; H.5.2 [Interfaces and Presentation]: User Interfaces—

1 Introduction

Multimodal interfaces often provide a more natural means of interaction [11, 24] and increase flexibility and reliability [1, 5]. Specifically, they provide promising alternative interaction techniques for Real-time Interactive Systems (RISs) in Augmented, Mixed, and Virtual Reality (AR, MR, and VR) [14], where the application of classical metaphors and input devices often is restricted or not possible at all.

Today, several frameworks support the development of multimodal interfaces for various application areas [1, 11, 19, 21, 23, 24]. Most of them foster maintainability, i.e. reusability, modifiability, or modularity [12], to counter the negative impacts of ad-hoc tailored application-specific solutions. However, their application to the RIS area is not as straightforward as one would expect. This issue is even further exacerbated, if such systems incorporate artificial intelligence (AI) methods, as often required for the implementation of multimodal interfaces [7, 14, 16]. Such intelligent RISs require a close temporal as well as semantic coupling of many multimodal processing steps. This is due to the fact that utterances are strongly related to the dynamically changing state of the virtual environment, perceived by the user, and the necessity of a common ground to resolve gestural and verbal expressions. The close coupling requirements call for alternative software techniques for multimodal processing frameworks in order to satisfy the general decoupling principle of frameworks.

This article focuses on the issue of semantic coupling and uses the following example interaction to highlight the proposed solutions. A user furnishes a virtual room in an architectural modeling application. She wants to relocate a particular green lamp from one table to another and utters:

> "*Grab* [pointing] *that green device. Put it on* [pointing] *that table*."

Our example is motivated by typical *put-that-there* interactions as introduced by Bolt [2]. The example is realized for (I_1) a fully immersive virtual reality application, where *pointing* signifies a natural pointing gesture performed with the user's whole arm and for (I_2) a mixed reality application on an interactive surface, where *pointing* signifies the user's finger touching a certain point on the interactive surface. Here, the semantic coupling is as follows: Processing of the first speech token will require to (1) check if there is an executable action that is associated with semantic concept *grab*. Thereafter, the processing of the first pointing gesture will require to resolve what objects can be found in the pointing direction, (2) are a device, and (3) are colored green. In addition, it is required to (4) check if the referent is a valid parameter for the corresponding action, i.e. if it is movable. Likewise, the second sentence has to be processed. However, the application state has to reflect the selection of the lamp in the first sentence, in order to allow the resolution of the pronoun *it*.

2 Related Work

Multimodal processing frameworks provide different sets of methods for multimodal processing and fusion at data-, feature-, and decision-level. Moreover, they utilize high-level graphical [19, 23] or text-based [1, 11, 14, 24] languages in order to increase API usability [3, 4], e.g., programmability or readability [5]. Generally, these frameworks foster maintainability by utilizing software engineering methods like modularization and composition [11, 23], component models [21, 24], or explicit interface definitions [14].

However, current application-independent frameworks struggle to provide a unified access scheme to the application state. This missing interface is a potential source for idiosyncratic implementations with poor reusability, modifiability, and modularity. While this issue is negligible for low-interactive application areas, it is a major issue for intelligent RISs. Multimodal frameworks that explicitly target intelligent RISs, like Latoschik's framework comprising *PrOSA* (Patterns On Sequences of Attributes) and *tATN*s (temporal Augmented Transition Networks) [14], typically provide a unified access scheme to the application state. However, such frameworks are often closely coupled to the underlying RIS platform and its main modules. In the case of [14], the unified access to a central Knowledge Representation Layer (KRL) proved to be of great benefit, e.g., for resolving pointing gestures. However, its heavy dependency on a proprietary scene graph hindered the long-lasting reusability of the framework. This coupling dilemma is a prominent problem in the RIS area [18]. In addition to those issues, a complete evaluation of a multimodal processing framework

*e-mail: martin.fischbach@uni-wuerzburg.de
†e-mail: dennis.wiebusch@uni-wuerzburg.de
‡e-mail: marc.latoschik@uni-wuerzburg.de

```
1  val Spine        = EntityType(spine)
2  val RightHand    = EntityType(hand)  and Chirality(Right) and Reference(sensor)
3  val RightHandRel = EntityType(hand)  and Chirality(Right) and Reference(spine)
4
5  Start a new Processor { //that
6   Requires property Transformation from RightHand
7   Requires property Transformation from Spine
8   Creates entity `with` properties RightHandRel
9   Updates the Transformation of entity RightHandRel `with`{
10   (Transformation of RightHand) relativeTo (Transformation of Spine)
11 }}
```

Listing 1: DSL-supported definition of semantic entity references by means of an enumeration of semantic values (lines 1-3). Semantic values are created by combining a semantic type (blue) with an appropriate value, in this case a semantic concept (brown). In addition, DSL-supported definition of a new thread of execution (Processor) that calculates the position of the user's left relative to her spine (lines 5-11): Required properties are described using semantic types and semantic entity references (purple, lines 6-7), a new sink entity is created (line 8), and a semantic function (orange) is used to define the calculation rule (lines 9-10). Since the DSL is defined using a native programming language, it can exploit sophisticated IDE features like syntax checks or auto-completion.

should ideally include (1) performance measures, (2) the evaluation of its API usability, as well as (3) a quality evaluation of supported processing methods. While (1) is paramount for RIS applications, (2) and (3) apply to all application areas. In some cases, accompanying performance analyses [11, 21] or developer and end-user usability evaluations [23] are reported. Beyond, the majority of contributions presents proof-of-concept applications as validation, leaving critical framework properties and potentially conducted evaluations unclear.

To our best knowledge there are no contributions that focus on the comprehensive evaluation or comparison of alternative multimodal processing techniques. We claim that this is at least partly due to the software quality problems discussed so far, especially involving a missing reusable unified access scheme.

Altogether, the area of multimodal processing for RIS currently lacks a platform that provides unified access to a global application state and at the same time sufficiently fosters maintainability. Such a platform would additionally provide an important step towards pending evaluations of existent multimodal processing techniques.

In the next sections, we will present two key solutions for the motivated issues: (1) a semantics-based access scheme to a globally accessible application state and (2) the specification of effects by means of *semantic functions*, which are suitable for multimodal processing. In contrast to work of others, e.g., Latoschik's FESN [17], we use standardized tools to further increase reusability. Moreover, our semantic description is directly connected to first-class citizens of the target programming language via a code generation approach.

3 Semantic-based State Access

Our globally accessible application state involves the adoption of an entity model, a widely accepted approach to represent a RIS application state [22]. Therein, *semantic entities* are represented as a set of properties which are described by *semantic types*, as presented by [28]. Such semantic types consist of a corresponding data type and a semantically represented concept, wherefore they provide basic support for semantic reflection [15]. Both the type and the concept are specified in a Web Ontology Language (OWL) ontology, which can be split over multiple files. Thereby, a modular architecture that comprises core concept files as well as (reusable) application specific ones is facilitated.

Type and concept definitions are supposed to be automatically transformed into respective code representations in the target programming language, allowing for their utilization in an eventual framework (see figure 1). In contrast to our previous work on semantic-based access schemes [26], which focuses on creating and relating entities as well as on observing and modifying properties of those entities, the approach presented in this paper deals with selecting entities from the global state using semantic descriptions.

We recall the interaction example I_1 from the introduction to illustrate the presented concepts: the user of a fully immersive virtual environment points at a green lamp and utters her desire to put it onto a table. In this context, e.g., the lamp is represented by a semantic entity, which (at least) contains a color and a transformation property (shown in the upper parts of figures 1 and 2).

Upon their creation, semantic entities are inserted into a central registry, which also is illustrated in the upper part of figure 2. In order to access existing entities—which constitute the entire application state—the central registry can be queried by passing an *entity filter*. Entity filters can process properties of semantic entities, check the existence of properties, or even utilize concepts like *semantic traits* (as suggested by [28]) to decide if a semantic entity meets the filter's criteria. The central registry answers such a request by returning all entities that match the filter.

The utilization of entity filters for multimodal processing is exemplified in listing 1: semantic entity references for the user's spine and right hand are defined in lines 1–3. Those references are used to create respective entity filters and query the central registry in lines 6 and 7. The results are then used by the thread of execution that realizes the defined processing step (called a *Processor*).

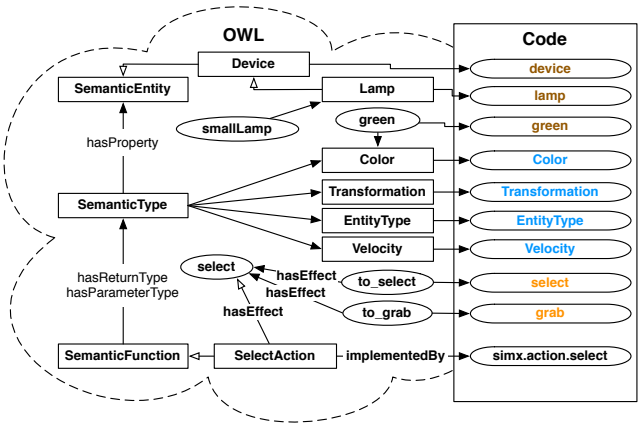

Figure 1: A conceptual overview of ontology contents and generated code elements: semantic concepts (brown), semantic types (blue), and semantic functions (green).

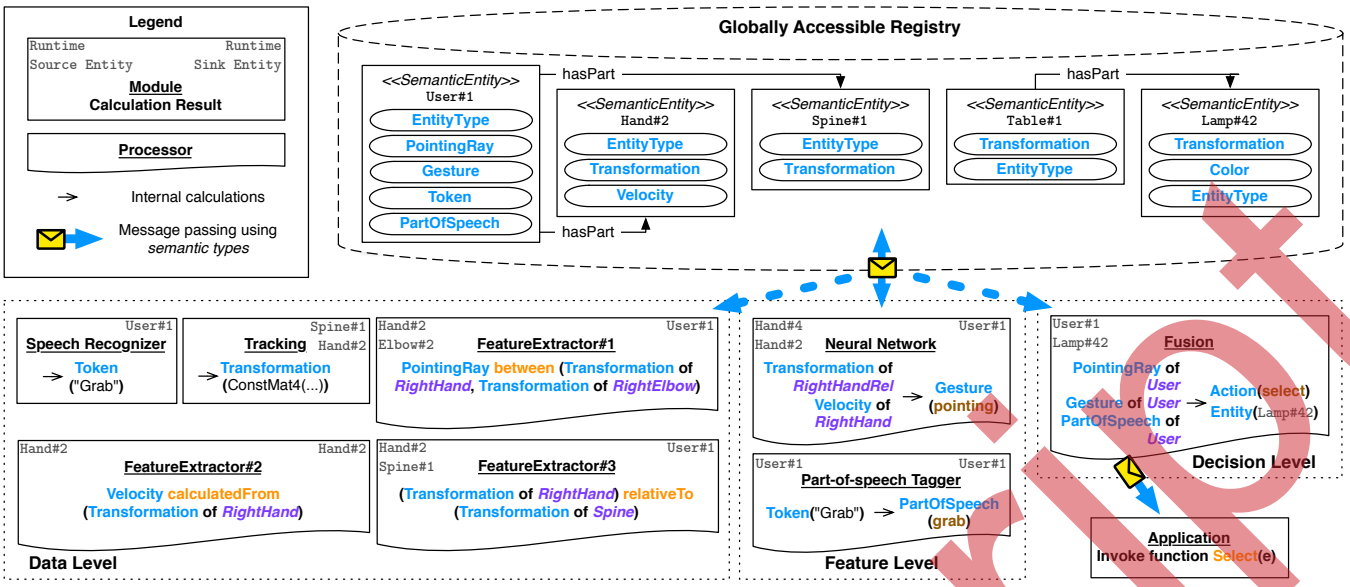

Figure 2: A conceptual overview of the utilization of the presented semantics-based techniques throughout the first sentence of example I_1: At data level, sensor data is mapped to semantic values (blue with subsequent value in braces) and associated with properties of semantic entities. Moreover, additional features are extracted from the tracking data using semantic functions (orange). At feature level, higher level symbolic properties, i.e. semantic concepts (brown), are derived. The results are fused at decision level and passed to the application. All levels use semantic types (blue), semantic values and semantic entity references (purple) to define requirements and provide their results to the application state by updating the associated semantic entity property.

4 Semantic Functions

In order to support a uniform definition of actions and processing steps, we propose an approach similar to semantic entities for re-occurring application- or domain specific functions: *semantic functions*. Their semantics and signature are described in the system's external OWL ontology, using semantic types for parameters and results. The respective interfaces for their application and implementation are created in the above-mentioned process of automatic code generation from that ontology. This way, the utilization of a semantic function removes implementation details from the description of a calculation rule or action sequence, while preserving (semantic) type safety. Semantic functions typically implement low-level computation methods to process (sensor) data as well as high-level actions that modify the application state.

In our example, low-level functions are used to process sensor data in order to extract features suitable to improve the pointing gesture detection, whereas the high-level `selection` function visually highlights a semantic entity: the velocity of the user's hand and its position relative to the spine are computed from data provided by the utilized tracking client. In the process, the concrete velocity value is obtained by invoking the respective semantic type's `calculatedFrom` function and accessing the hand's transformation using a semantic type (see figure 2, *FeatureExtractor#2*). Similarly, the relative position is computed using the `relativeTo` function (see listing 1 and figure 2, *FeatureExtractor#3*).

Computed properties are stored in entities and accessed via the KRL by other modules (shown in the lower right of figure 2): velocities, transformations, and relative positions are fed into a neural network that is able to detect gestures. Detection results are inserted into the respective entity property and subsequently available via the associated semantic type `Gesture`. Similarly, the part-of-speech tagger maps detected tokens to semantic concepts, e.g., the word "grab" to the concept `grab`. Verbs denote functions, whereas adjectives represent properties and nouns represent entities. The associated semantic functions, semantic entities, and their properties are accessed, utilizing the ontological grounding. For example, the adjective "green" is identified to be an instance of the semantic type `Color` and "grab" is detected to denote a `SelectAction` (cf. figure 1).

With this information, the fusion module determines the entity referenced by the user: the central registry is queried using an entity filter that includes all user-specified and semantically described features. Finally, the application executes the previously determined semantic function `select`, passing the identified entity.

5 Implementation

The implementation of the presented concepts is realized within the Simulator X framework [13], an open source research platform for RIS applications. It already provides beneficial aspects regarding maintainability, which especially include an entity model and an architecture that is built on the actor model [10]. This entity model is composed of semantic entities that are realized by means of semantic types. Moreover, semantic functions are utilized to define actions and processing steps.

Both, semantic types and semantic functions are defined in an OWL ontology and automatically transformed into the the used programming language Scala (see figure 3). In addition to the runtime references of semantic types, -values, and -functions to the application's ontology, the entity model facilitates the representation of relations between entities (for details see listings 6, 7, and 8 in the appendix). Dedicated reasoning software, like HermiT or Pellet, can thus be used to infer new information about semantic entities and add inferred information to the application's state. In the same way, information about the current state of the simulation can be added to the ontology and in turn be used by the reasoning software.

The presented techniques extend the Simulator X platform and constitute essential building blocks of the Multimodal Input PROcessing framework *miPro* [16]. miPro is built upon Simulator X and provides a uniform description language for interconnecting

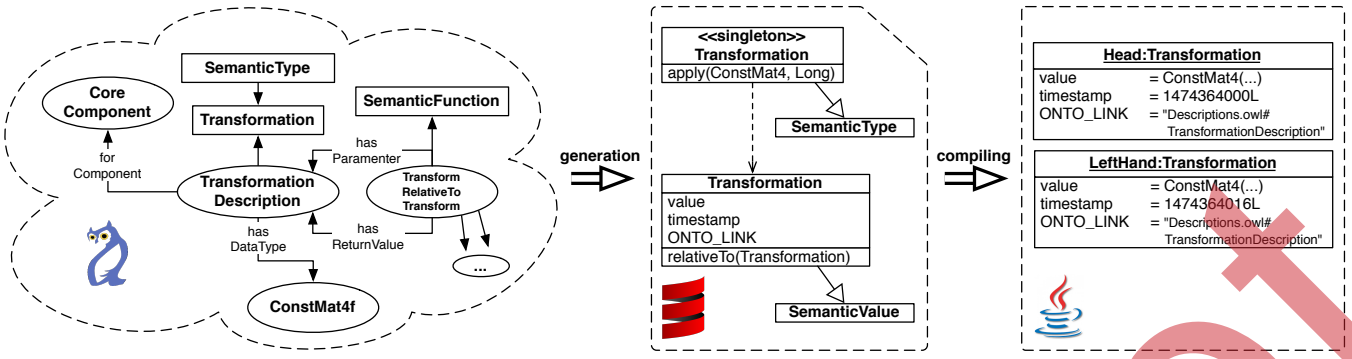

Figure 3: A an overview of the transformation of OWL definitions into Scala code and their usage: Semantic types and semantic functions are defined in an OWL ontology (see listings 2 and 3 in the appendix for details). Before the actual implementation, these definitions are used to automatically generate respective Scala first-class citizens (see listing 4 in the appendix for details). During the implementation phase, semantic types can be combined with actual values to create new instances of so called *semantic values* (see listing 5 in the appendix for details), to query semantic entity properties, or to apply semantic functions (cf. listing 1). During runtime on the Java Virtual Machine (JVM), semantic types, -values, and -functions still have a reference to the application's OWL ontology.

multimodal processing steps, as showcased in listing 1. Using Simulator X and miPro results in multiple benefits, as discussed below.

Simulator X is implemented using the programming language Scala. Due to the uniform definition of semantic types and functions as well as Scala's capability to omit the dot operator in certain cases, it is feasible to create convenient Domain Specific Languages (DSLs) that foster API usability. This feature is especially utilized by the description language of the miPro framework.

The benefits of a generative approach regarding maintainability include the reduction of potentially misused identifiers, the incorporation of a reference to the concepts' ontological definition, the resulting connection to relations to other concepts, and the possibility to specify knowledge about application content outside of the actual program code.

Moreover, the set of all registered entities and the described semantics-based access constitute a core-level KRL. Its architecture is similar to a blackboard model [20]. However, there is no multi-agent system in our framework, which is concurrently working on the blackboard. In addition, we do not use high-level messages like KIF/KQML [6], reducing marshaling overhead.

The semantics-based approach decouples the process of accessing the application state from the actual implementation of the RIS framework. This way, application developers can create software without knowing about the exact composition of software components, e.g., in terms of used classes and data types. Thus, changes to this composition that retain its feature set do not affect an application. Therefore, development, integration, and replacement of framework components is immensely simplified.

Altogether, the implementation allows for maintaining the functionalities that have been created for the interaction example I_1 and to reuse them in the second example I_2: Therein a pointing gesture would not be detected by a neural network but directly extracted from the touch on the interactive surface. Due to the coupling on a semantic level, all other parts of the application can be retained. Similarly, an alternative implementation of the `select` function can be provided, which fits the new application's needs. On a higher level, complete simulation modules can be easily exchanged. For example, the 3D rendering component used for I_1 can be replaced by a more appropriate 2D renderer.

6 Results

The presented semantics-based techniques support maintainability in various facets: Types and functions are decoupled by means of semantic concepts. This eases their reuse in different multimodal applications. For instance, the `select` function is useful in many applications and can be utilized with minimal effort. The same is true for types like color or transformation. In addition, the approach of generating semantic types and semantic functions from an ontology fosters reusability.

Existing reasoning modules can be (re)used to query the central registry, e.g., to be able to handle the utterance "Grab [pointing] that green light" by matching *light* with the semantic concept *lamp*. Moreover, the definitions of semantic entities and semantic functions become independent of the target programming language. They thus can be reused for many different RIS platforms.

Our implementation of these concepts on top of *Simulator X* furthermore supports the reuse of many processing steps even in different application areas with divergent module configurations. This is due to decoupling, achieved by the core-level integration of semantics-based software techniques; a unique characteristic of our approach.

Semantic types and semantic functions are also beneficial in terms of flexibility, since all specifications are based on semantic concepts—not data types. This allows to better cope with changes or diversity of underlying data types. The representation of a transformation or color, for example, can be altered requiring no changes in high-level DSL descriptions and only minor changes in the data-type related low-level (function) implementations. The required effort can be additionally reduced using automatic type converters, as proposed by [27]. Moreover, semantic functions can be easily adapted in the ontology, e.g., to accept alternative verbs for a selection task. Likewise, semantic types and semantic functions are extended by adding new elements to the respective OWL file. For those use cases, we developed a plugin for the protégé OWL editor that extends existing editing capabilities.

All our software components provide an OWL file, which imports a set of basic concept definitions that is shared by all components. These shared concepts comprise basic semantic types, for example position, rotation, color, and simple relations (e.g., has-part). Each software component adds specific concepts, which are relevant for its functionality. Finally, each application combines the files of the used components and addditional domain specific concepts, resulting in the applications knowledge base. A possible structure for such ontologies, which is currently applied by the open source Simulator X framework, is described in [25].

We believe that, in the long term, ontology files with common basic concepts as well as files that are shared between components of the same type (e.g., renderers or physics engines) will evolve. Al-

though there will most probably be a certain part of the domain specific ontology that is specific to one application, other parts could be split into multiple reusable files.

7 Conclusion

The presented semantics-based access to a globally accesible application state and semantic functions are general software techniques to address the requirement of close semantic coupling. They support maintainability even under the challenging conditions of the RIS domain. Due to their intrinsic semantic grounding they are especially suitable for intelligent RISs, as required for the implementation of multimodal interfaces. They provide a valid solution for the coupling dilemma and are also applicable for the definition of application logic.

The evaluation of maintainability beyond expert reviews is an important target of our ongoing research, which is no easy endeavor due to the complexity of the systems. First static as well as dynamic code analyses have been conducted for Simulator X [25]. Explorations and proof of concept evaluations have been done in the scope of various VR and MR projects [8, 9, 29]. In addition, we use the framework for practical parts of lecture modules. *Simulator X* and *miPro* are available to the public to enable feedback for maintainability refinement and to promote reuse.

References

[1] A. Ameri Ekhtiarabadi, B. Akan, B. Çürüklu, and L. Asplund. A General Framework for Incremental Processing of Multimodal Inputs. In *Proceedings of the 13th International Conference on Multimodal Interfaces*, ICMI '11, pages 225–228. ACM, 2011.

[2] R. A. Bolt. „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, pages 262–270, New York, NY, USA, 1980. ACM.

[3] S. Clarke. Measuring API usability. *Dr. Dobb's Journal*, 2004.

[4] J. M. Daughtry, U. Farooq, J. Stylos, and B. A. Myers. Api usability: Chi'2009 special interest group meeting. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '09, pages 2771–2774. ACM, 2009.

[5] B. Dumas, D. Lalanne, and S. Oviatt. Multimodal interfaces: A survey of principles, models and frameworks. In *Human Machine Interaction*, pages 3–26. Springer, 2009.

[6] T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, S. Shapiro, J. McGuire, R. Pelavin, and C. Beck. Specification of the kqml agent-communication language. 1994.

[7] M. Fischbach. Software techniques for multimodal input processing in realtime interactive systems. In *Proceedings of the 17th International Conference on Multimodal Interaction (ICMI '15)*, ICMI'15. ACM, 2015.

[8] M. Fischbach, D. Wiebusch, A. Giebler-Schubert, M. E. Latoschik, S. Rehfeld, and H. Tramberend. SiXton's curse - Simulator X demonstration. In *Virtual Reality Conference (VR), 2011 IEEE*, pages 255–256, 2011.

[9] M. Fischbach, D. Wiebusch, M. E. Latoschik, G. Bruder, and F. Steinicke. Blending Real and Virtual Worlds Using Self-reflection and Fiducials. In *ICEC*, volume 7522 of *Lecture Notes in Computer Science*, pages 465–468. Springer, 2012.

[10] C. Hewitt, P. Bishop, and R. Steiger. A Universal Modular ACTOR Formalism for Artificial Intelligence. In *Proceedings of the International Joint Conference on Artificial Intelligence*, IJCAI'73, pages 235–245. Morgan Kaufmann Publishers Inc., 1973.

[11] L. Hoste, B. Dumas, and B. Signer. Mudra: A Unified Multimodal Interaction Framework. In *Proceedings of the 13th International Conference on Multimodal Interfaces*, ICMI '11, pages 97–104. ACM, 2011.

[12] ISO. Systems and software engineering – systems and software quality requirements and evaluation (square) – system and software quality models. ISO ISO/IEC 25010:2011, International Organization for Standardization, Geneva, Switzerland, 2011.

[13] M. Latoschik and H. Tramberend. Simulator X: A scalable and concurrent architecture for intelligent realtime interactive systems. In *Virtual Reality Conference (VR), 2011 IEEE*, pages 171–174, 2011.

[14] M. E. Latoschik. A User Interface Framework for Multimodal VR Interactions. In *Proceedings of the 7th International Conference on Multimodal Interfaces*, ICMI '05, pages 76–83. ACM, 2005.

[15] M. E. Latoschik. Semantic reflection–knowledge based design of intelligent simulation environments. *KI 2007: Advances in Artificial Intelligence*, pages 481–484, 2007.

[16] M. E. Latoschik and M. Fischbach. Engineering Variance: Software Techniques for Scalable, Customizable, and Reusable Multimodal Processing. In *Proceedings of the HCI International Conference*, pages 308–319. Springer, 2014.

[17] M. E. Latoschik and M. Schilling. Incorporating VR databases into AI knowledge representations: A framework for intelligent graphics applications. In *Proceedings of the Sixth IASTED International Conference on Computer Graphics and Imaging*, pages 79–84, 2003.

[18] M. E. Latoschik and H. Tramberend. Short Paper: Engineering Realtime Interactive Systems: Coupling & Cohesion of Architecture Mechanisms. In *Proceedings of the 16th Eurographics Conference on Virtual Environments & Second Joint Virtual Reality*, EGVE - JVRC'10, pages 25–28. Eurographics Association, 2010.

[19] J.-Y. L. Lawson, A.-A. Al-Akkad, J. Vanderdonckt, and B. Macq. An Open Source Workbench for Prototyping Multimodal Interactions Based on Off-the-shelf Heterogeneous Components. In *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '09, pages 245–254. ACM, 2009.

[20] H. Nii. The blackboard model of problem solving and the evolution of blackboard architectures. *AI magazine*, 7(2):38, 1986.

[21] M. Serrano, L. Nigay, J.-Y. L. Lawson, A. Ramsay, R. Murray-Smith, and S. Denef. The Openinterface Framework: A Tool for Multimodal Interaction. In *Extended Abstracts on Human Factors in Computing Systems*, CHI EA, pages 3501–3506. ACM, 2008.

[22] A. Steed. Some Useful Abstractions for Re-Usable Virtual Environment Platforms. In M. E. Latoschik, D. Reiners, R. Blach, P. Figueroa, and R. Dachselt, editors, *IEEE VR Workshop on Software Engineering and Architectures for Realtime Interactive Systems*, pages 33–36. Shaker Verlag, 2008.

[23] W. W. Tang, K. W. Lo, A. T. Chan, S. Chan, H. V. Leong, and G. Ngai. I*Chameleon: A Scalable and Extensible Framework for Multimodal Interaction. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 305–310. ACM, 2011.

[24] J. Wagner, F. Lingenfelser, T. Baur, I. Damian, F. Kistler, and E. André. The Social Signal Interpretation (SSI) Framework: Multimodal Signal Processing and Recognition in Real-time. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13, pages 831–834. ACM, 2013.

[25] D. Wiebusch. *Reusability for Intelligent Realtime Interactive Systems*. PhD thesis, Universität Würzburg, 2015.

[26] D. Wiebusch and M. Latoschik. A uniform semantic-based access model for realtime interactive systems. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2014 IEEE 7th Workshop on*, pages 51–58, March 2014.

[27] D. Wiebusch and M. E. Latoschik. Enhanced Decoupling of Components in Intelligent Realtime Interactive Systems using Ontologies. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, pages 43–51. IEEE, 2012.

[28] D. Wiebusch and M. E. Latoschik. Decoupling the Entity-Component-System Pattern using Semantic Traits for Reusable Realtime Interactive Systems. In *IEEE VR Workshop on Software Engineering and Architectures for Realtime Interactive Systems*, IEEE VR, 2015.

[29] C. Zimmerer, M. Fischbach, and M. E. Latoschik. Fusion of mixed reality tabletop and location-based applications for pervasive games. In *Proceedings of the 2014 ACM International Conference on Interactive Tabletops and Surfaces*. ACM, 2014.

8 Appendix

```
<Descriptions.owl#TransformationDescription>
    rdf:type <CoreOntology.owl#SemanticType> ,
    <Types.owl#Transformation> ,
    owl:NamedIndividual ;
  <CoreOntology.owl#forComponent>
    <CoreComponent.owl#CoreComponent> ;
  <CoreOntology.owl#hasDataType>
    <Types.owl#simplex3d.math.floatx.ConstMat4f> .
```

Listing 2: Simplified Turtle (Terse RDF Triple Language) definition of the semantic type `Transformation` used by the Simulator X platform.

```
<Functions.owl#TransformRelativeToTransform>
    rdf:type <CoreOntology.owl#SemanticFunction> ,
    owl:NamedIndividual ;
  <Functions.owl#hasName>
    "relativeTo"^^xsd:string ;
  <Functions.owl#implementedBy>
    "functions.Default.realtiveTo"^^xsd:string ;
  <Functions.owl#hasOperatorType>
    <Functions.owl#InfixOperatorType> ;
  <Functions.owl#hasParameter>
    <Descriptions.owl#TransformationDescription> ;
  <Functions.owl#hasReturnValue>
    <Descriptions.owl#TransformationDescription> ;
  <Functions.owl#hasParameter>
    <Descriptions.owl#TransformationDescription> .
```

Listing 3: Simplified Turtle (Terse RDF Triple Language) definition of the semantic function `relativeTo` used by the Simulator X platform.

```
object Transformation
  extends SemanticType[ConstMat4f](
    types.NullType as
    SemanticConcepts.transformation withType
    classOf[ConstMat4f] definedAt
    "Types.owl#Transformation"
  )
{
  override def apply(
    value: ConstMat4f,
    timestamp : scala.Long
  ): Transformation =
    new Transformation(value, timestamp)
}

class Transformation(
  value : ConstMat4f,
  timestamp: Long
  ) extends SemanticValue[ConstMat4f](
    value,
    timestamp,
    Transformation
  )
{
  def relativeTo(newCs : Transformation)
                (implicit functions: Functions) =
    functions.relativeTo(this, newCs)
}
```

Listing 4: Simplified generation output for the semantic type `Transformation` showcased in listing 2 and the semantic function showcased in listing 3.

```
def setTransformation(t: ConstMat4f, e: Entity) {
  val semanticValue = Transformation(t)
  e.set(semanticValue)
}
```

Listing 5: Example usage of the generated scala class and -object showcased in listing 4: The apply method of the `Transformation` object (semantic type) is called by passing a `ConstMat4f` to create a new respective semantic value instance (line 2). This instance is used to set a property of an entity (line 3).

```
<Types.owl#hasPart>
  rdf:type owl:ObjectProperty ;
rdfs:subPropertyOf
  <CoreOntology.owl#SimX_BinaryRelation> .
```

Listing 6: Simplified Turtle (Terse RDF Triple Language) definition of the relation `HasPart` used by the Simulator X platform.

```
class RelationType
  extends SemanticType[Relation]
{/*...*/}

object HasPart
  extends RelationType(
    types.Relation as
    SemanticConcepts.hasPart definedAt
    "Types.owl#hasPart"
  )
{/*...*/}
```

Listing 7: Simplified generation output for the relation `HasPart` showcased in listing 6.

```
def connect(user: Entity, arm: Entity) {
  user.set(types.HasPart(arm))
}
```

Listing 8: Example usage of the generated scala object showcased in listing 7: The apply method of the `HasPart` object (relation) is called by passing an entity (line 2). The `set` method of the `entity` class is overloaded for relations. It internally creates a `Relation` instance, which holds the subject and object of a relation (in this case `user` and `arm`), uses it to create a respective semantic value, and finally calls it's `set` that takes a semantic value.