THEORETICAL UNDERPINNING AND PROTOTYPE IMPLEMENTATION OF SCENARIO BUNDLE-BASED LOGICAL CONTROL FOR SIMULATION OF HUMAN-ARTIFACT INTERACTION

Wilhelm Frederik van der Vegte
Corresponding author
Faculty of Industrial Design Engineering
Delft University of Technology
Landbergstraat 15, 2628 CE, Delft, the Netherlands
w.f.vandervegte@tudelft.nl
telephone +31-152781061, fax +31-152781839

Imre Horváth
Faculty of Industrial Design Engineering
Delft University of Technology
the Netherlands
i.horvath@ tudelft.nl

ABSTRACT

This article presents a new methodology that enables designers to include in simulations not only the physics aspects of artifact behavior, but also human actions. The motivation for this research came from the fact that none of the conventional approaches to engineering simulations includes manipulative control of products by users as foreseen by designers. By implementing control over physics simulations, changes in parameters can be introduced that alter the course of the simulated process. As a means to do this, we propose to use scenario bundles, with which designers can operationalize their conjectures of how human users interact with products as a series of interconnected simulations. For the imaginary use process described in a scenario bundle, the designer can specify various product designs, user characteristics and environments, which may in each case lead to different concatenations of simulation actions. The proposal facilitates the exploration of possible mismatches and anomalies in use processes. In this article we have described the theoretical fundamentals and the overall concept of the proposed methodology, as well as its realization as a proof-of-concept implementation. This implementation can be used as a tool to specify scenario bundles and to perform controlled simulations of human-product interaction. The use of the tool is demonstrated through a practical example. Although the implementation has proven to be successful in terms of executing scenario bundles, two bottlenecks need further attention: (i) devising stable algorithms for large deformations in physical interaction simulation and (ii) incorporation of already existing algorithms for simulation of low-level human motion control.

KEYWORDS

Scenario bundles, product design, virtual prototyping, simulation control, use process, human-product interaction

1 INTRODUCTION

An important goal of user-centered design is designing consumer durables for optimal interaction with users, i.e., considering how products can be used by various users in various situations. This can be facilitated through computer tools that afford concurrent simulations of the physical behavioral processes as well as the interaction processes. While a large number of multiphysics-based behavior simulation tools have been developed, less advancement has been achieved in the field of simulating human-product interaction under varying circumstances. Typically, possible forms of use are investigated through usability testing, i.e., with human subjects and physical prototypes, but this takes a lot of time and resources [1]. Those currently available computer-aided approaches that

do *not* require physical prototypes and human subjects fail to offer product designers intuitive means for specifying and controlling behavioral and interaction simulations [2]. The work described in this article aims to close this gap and to offer a solution for the problem described above.

The goal of our research was to develop the theoretical fundamentals and a comprehensive concept for simulating alternative product use scenarios including foreseen contacts and actions. In addition, our objective was also to convert the concept of scenario-based simulation to a testable practical implementation. We wanted to make it possible for designers to 'play' with scenarios as explorative means. Figure 1 shows the key concepts of our approach and the activities the designer is supposed to perform with the envisaged system. Presentation of the underpinning theory and the computational implementation of the tool forms the core of this article. An initial version of this theory was specifically developed to create a basis for the integration of physics simulations with logical control of *nucleus-based models* (see [3, 4]). This was previously published in a PhD thesis [5]. For this article the theory has been generalized and now does not depend on any particular modeling and simulation method, i.e., the simulation model can be based on finite-element models, block diagrams, bond graphs, etc.

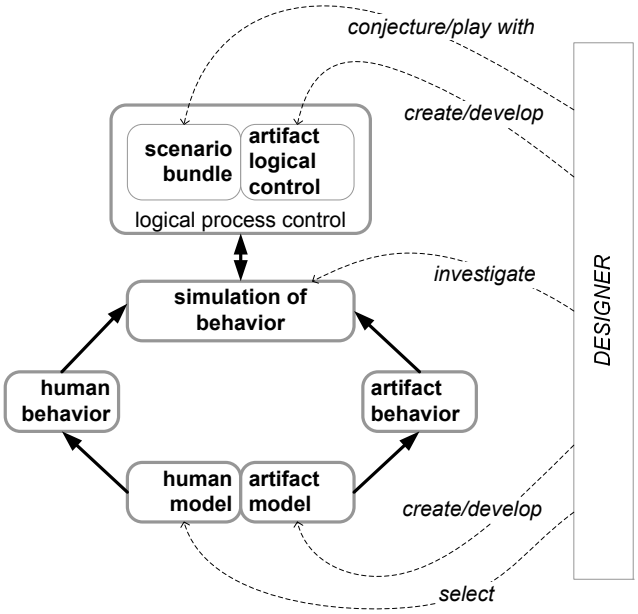


Figure 1. Key concepts of the approach presented in this article

In the next section, we briefly survey related work in the field of human-artifact interaction simulation. In Section 3, we present our concept for controlled interaction simulation with scenarios. This concept is further formalized in Section 4 to facilitate its computational implementation. In Section 5 we will present a proof-of-concept implementation that we have realized using commercially available software. In Section 6, we will describe an application example and discuss the concrete and specific models we created. The theory, its implementation, and further application opportunities will be discussed in Section 7, followed by conclusions and plans for future work in Section 8.

2 RELATED WORK

With special attention to conceptual design of consumer durables, we propose a novel control mechanism that allows simulation of user-product interaction in envisaged use contexts. The mechanism allows investigation of multiple use scenarios in these contexts using *scenario bundles*. When implemented in a software tool, the mechanism enables monitoring the actions and history of interaction processes, alternative considerations of use actions by designers, and optimization of the interaction from the perspective of the users and the products. A scenario bundle formally describes a set of alternative use scenarios of a product that includes the foreseen physical contacts and actions. Technically, it means that we apply logical control over simulations to specify specific interaction sub-processes.

The importance of the proposed approach comes from the fact that scenario-based simulations can provide designers with quantitative feedback on complete interaction sequences. This is an improvement over the functionality offered by conventional engineering simulation tools, which typically require dedicated simulation

runs for each specific episode of the use process. Complete interaction sequences are supposed to provide designers better clues to improve products with a view to interactions. By constructing and testing a scenario bundle, a designer can perform 'what-if' type of investigations involving various product designs, physical properties, use environments, and human users. When the arrangement is changed, the interaction simulation may take a different course through the scenario bundle. Additionally, scenario-based simulation can be a low-threshold alternative to testing of physical prototypes or conducting interactive (participatory, human-in-the-loop) simulations, since there is no need to employ human subjects.

Both in the scientific literature and on the market of commercialized software packages, several methods and applications can be found that claim to offer designers simulations of interactions between humans and artifacts in the virtual realm. Two primary application fields can be distinguished, namely, entertainment (movies and gaming) and ergonomics evaluation of products and systems. In both areas the level of sophistication of an approach is typically assessed based on two criteria: (i) physical correctness (i.e., does the simulation respect the laws of physics?) and (ii) naturalness (in particular, of movements). In both cases, physical correctness is typically ensured by using the laws of physics as a basis for the simulation algorithms. However, in entertainment applications, the laws of physics are typically selectively applied (e.g., to physically constrained motions) in order to save computation time [6]. In ergonomics evaluation, where for instance forces on joints have to be monitored [7], physical correctness is typically maintained at all times.

Naturalness in entertainment applications implies that movements *look* natural, and this aspect is typically visually evaluated with human subjects [6]. In ergonomics evaluation however, it is important that movements correspond to how a real human would move. Therefore, in those evaluation tools that are capable of dynamical movement simulation, movements are typically based on, or have been validated by comparison with, real human movements numerically quantified in the metric space [e.g., 8]. Since our approach aims at product evaluation, which includes ergonomics aspects, we have focused our review of related work on contributions from an ergonomics perspective where physical correctness and naturalness have been interpreted more strictly.

Conventional ergonomic manikins, such as Sammie, HECAD, and Combiman, can assume arbitrary postures in a virtual environment containing various artifacts. However, motions are not simulated but have to be completely directed by the system user, and forces are not computed [9]. Using commercial simulation packages such as LifeModeler [10] and Anybody [11], designers can bring together geometric models of humans and products to compute muscle forces and interaction forces during the use process. These software packages require predefined motion-capture frames obtained from sessions in which human subjects perform the task to be simulated. These predefined frames act as open-loop control over human motion, so that possible influences of the product on the human user cannot be taken into account.

The Jack manikin builds further upon the conventional ergonomic manikins and offers a library of predefined motion-frame sequences related to common tasks [12]. These can be sent to the manikin during interactive sessions controlled by the system user. Prediction of kinetic effects is limited to quasi-static calculation of forces on human joints. To predict typical interaction postures for conventional ergonomic manikins, Singh et al. proposed a constraint-based technique [13]. Given the position and orientation of one or more product surfaces to be gripped by the hands, the algorithm computes whole-body postures. This is one step towards elimination of guidance by the system user, but the algorithm lacks a mechanism to compute transitional motions between different gripping situations.

Santos is a virtual human manikin provided that can generate transitional motion patterns between pairs of arbitrary start and end postures [8, 14]. A similar approach was presented in [15]. These approaches have been validated

with human subjects and they allow kinematics and kinetics simulation of humans interacting with artifacts while going through pre-programmed successions of key postures, under the implicit condition that the posture change is not disturbed (e.g., by a collision with an artifact). What cannot be simulated with these optimization-based motion trajectory techniques is the succession of posture changes in a complete multiple-interaction use process. In other words, the simulations support only low-level human motion control between the postures but not the high-level control (human decision-making) that determines the intent of each subsequent intended posture [cf.,16], nor the matching kinematical configuration of the human body as the technique in [13] provides. Consequently the optimization-based techniques still require a human in the loop who directs the virtual human during simulation.

In the ACT-R/DHM[1] project, Santos has been enhanced with a *cognitive-architecture* (CA) simulation to substitute the human in the loop [17]. Developed and empirically validated by psychologists, CAs (such as ACT-R) are production-rule based blueprints of the operational structure of cognition [18]. Production rules have been prepared for specific interaction tasks with specific products, especially for use processes that involve typical frequently occurring tasks, in particular driving cars [19], piloting aircraft [20], and interacting with computers [21]. For other 'atypical' tasks related to the use of other products, new production rules need to developed and validated. This is a laborious endeavor, which may explain why the ACT-R/DHM researchers so far have only considered the use of one product, a vending machine, in their efforts to apply CAs to control of dynamical simulations with 3D human models [22].

An important limitation of CA simulations is that they predict the *time* the human brain needs for processing perceptual input and for directing motor operations, but not the *reasoning* that determines which operation is performed in which situation. CA-based simulations, including ACT-R/DHM, start from a given *task decomposition*, i.e., an idealized sequence of operations as it can be extracted from an instruction manual. However, in reality there is not just one idealized way to use a product; instead use often involves going through a series of choices from subsequently available options [23]. This can be depicted by a decision tree, through which the user moves from an initial state to the goal state, selecting between available operations. Various paths from each junction in the tree represent state-transforming operations, one of which is selected. Each of the possible routes connecting junctions is a *scenario of use* [24], and the idealized task decomposition that is used to control CA-simulations is just one scenario that ignores the logic of decision-making.

As far as the application of scenarios to use processes with multiple possible courses is concerned, there have been efforts to achieve logical control over physical and/or simulated processes. As a more flexible alternative to the scenario-tree representation, *organized sets of scenarios* or *scenario networks* have been put forward, that allow repetitive loops and other forms of branching [25],[26]. These have been typically implemented as control models by using finite automata (FA) representations. In software engineering, organized sets of scenarios are commonly used in requirements specification, verification, and prototyping [27]. Since software prototyping is usually done by executing or emulating the program under development, while physical interaction with hardware (mouse, keyboard, etc.) and also human processing time (as can be simulated with CAs) is usually not investigated, additional simulations are not needed. As representations for organized sets of scenarios, *Statecharts* [28] and, to a lesser extent, *Petri nets* [29] have become popular [25],[26],[30],[31]. Although FA representations can also be based on formal symbolic constructs and text, these graphical notations are generally considered easier to work with [32].

In domains other than software engineering, the majority of the scenario-based control approaches have been developed for computer animations in training, gaming, and entertainment rather than for virtual prototyping. In

[1] *adaptive control of thought-rational* combined with *digital human modeling.*

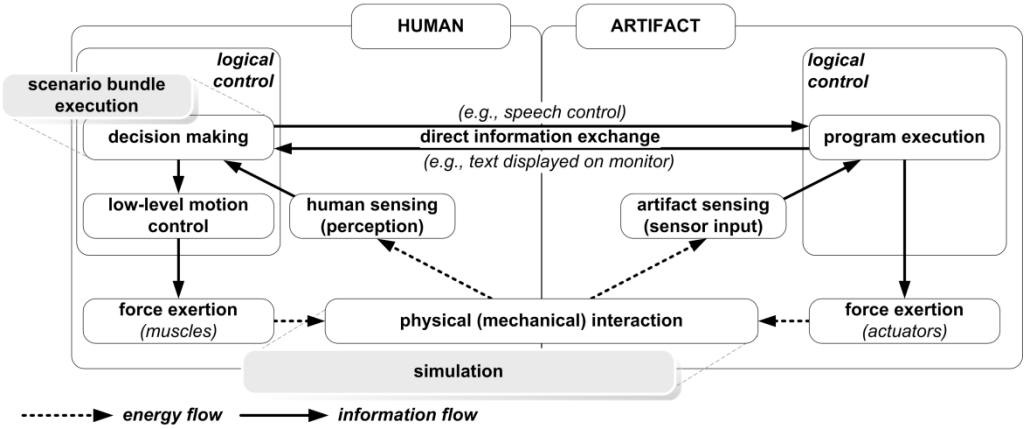


Figure 2. Conceptual reasoning model of human-artifact interaction

these approaches, movements are mostly generated based on pre-collected motion frames [33]. An exception is the Iowa driving simulator [34], which can project virtual humans driving around in virtual cars, and perform physics simulation controlled by scenarios specified in the form of statecharts. These statecharts cover both human decision-making and control embedded in artifact subsystems. Human motion was not simulated. Instead only its effects on the controls of the car (steering wheel, brake, etc.) were included.

In product design, application of scenarios as a means to control simulation of use processes with inclusion of human motor control and physical interaction, is rare. In the explorative phase of our research we could find only two references in the investigated literature on the application of scenarios. The first was in a side remark on possible future implementation in the Santos virtual human [14]. The other was in the article of Hou et al., who claimed to have implemented scenarios to simulate human-product interaction using a geometric/anatomical human model [35]. Unfortunately, implementation details are missing, and no reports on further developments after 2007 could be found.

We have concluded that in the most advanced systems developed for use-process simulations, no adequate link has been established between high-level and low-level human motion control, except in the cases where high-level control is performed by a human in the loop. The current approaches lack the opportunity to enable high-level control of human movements and interaction with artifacts by using scenarios. In the remainder of this article we will present a solution to operationalize formal scenarios in 'non-interactive interaction simulation', i.e., to simulate use situations without requiring a human in the loop to control the interactions.

3 INTRODUCING THE CONCEPTS AND FORMAL DEFINITIONS

3.1 General concept

Our idea has been that designers can investigate and optimize use processes of products if they have an appropriate mechanism to control simulation of the involved interactions and behavior of humans and artifacts. We adopted the concept of scenarios to implement the control mechanism. Figure 2 shows our reasoning model of human-artifact interaction. It brings together (i) high-level and low-level human control, with (ii) (optional) the control built into artifacts, i.e., recognition and detection by sensors, program execution by control mechanisms, and force exertion by actuators, and (iii) the actual physical (mechanical) interaction between the human and the product. As is indicated in the figure, a scenario bundle is a specification of human decision-making by the designer with respect

to this reasoning model. As in other fields where scenarios have been applied, such as HCI [cf. 26], scenarios in the bundle describe sequences of actions both by the human user and by the product. However, the scenario bundle does this *from a human perspective*, i.e., the human actions appear as actual actions, and the *effects* of actions performed by the product appear as achieved sub-goals of use that trigger the human to proceed to the next action. Achievement of sub-goals of use is expressed in the form of *events* and *conditions*.

The logical information processing acts as a control mechanism over simulation of structural and morphological changes, in which the physical interaction between humans and artifacts is manifested. Beside physical interaction, the simulation may cover any process that is governed by the laws of physics and captured in a mathematical model, such as, for instance, analog signal processing or heat exchange. To enable the control mechanism, it is interfaced with the simulation. The interfacing requires a two-way conversion: (i) of signals representing energy to an interpretation as information, and (ii) of logical instructions to physical variables that effectuate energy flows. The first conversion corresponds to sensing (perception in humans and sensor input to artifacts) and the second one to activation of effectors (muscles and actuators, respectively).

The components of the human-artifact system have been represented by particular *models, specifications,* and *constructs.* When describing behaviors, 'models' in our terminology refer to descriptions reflecting laws of nature and 'specifications' refers to executable conjectures about behavior or instructions. We have used the term 'construct' as a unifying notion that covers both 'models' and 'specifications' so as to address combinations of the two. For descriptions created by the designer (e.g., a 3D description of a product with material specifications), we have kept to the regular use of the word 'model'. Such models can also be subject to executed sequences of instructions, but in our case their use is limited to simulations based on laws of nature.

The following models, specifications, and constructs have been defined: (i) *logical constructs,* (ii) a *simulation model,* and (iii) a *signal conversion specification*. Logical constructs generate output that prescribes movements by humans and artifacts. This output takes the form of *control signals* that represent those parameters in the simulation model that specify motions of muscles and actuators. The simulation model describes the geometry and physical characteristics of humans and artifacts, and thereupon prescribes their behavior. The simulation layer outputs *meter signals* that describe selected/specified physical quantities in the simulation output that can be 'measured'. The signal conversion specification describes how the effects of behavior are 'sensed' by the logical constructs. The sensing is based on instantiation of *events* that mark the occurrence of specified change in meter signals. The three categories of representations and the related computational operations imply a stratified structure consisting of three layers, i.e., a modeling layer, *M*, a simulation layer, *Ψ*, and a logistics layer, *L* of which the processing scheme is shown in Figure 3. The focus of this article is on adding logical control to simulations. The other two layers are shown in dashed lines.

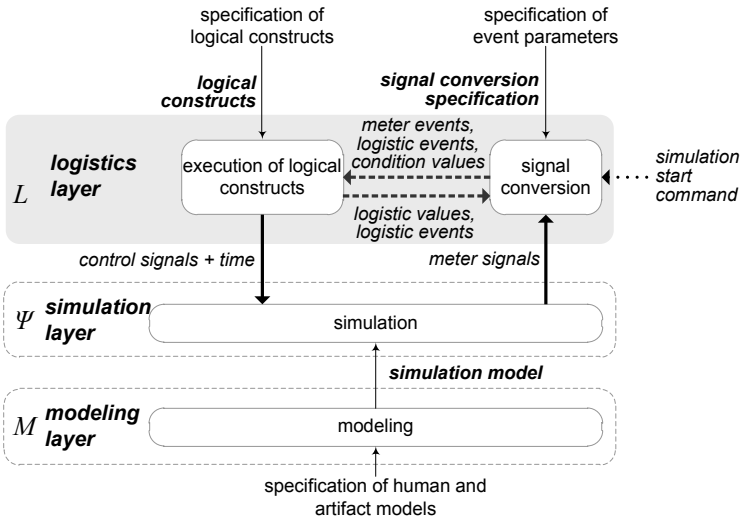


Figure 3. Processing scheme of three-layered modeling and simulation

Formally, the logistics layer can be described as $L = \{\Lambda, \Xi\}$ where Λ is the *set of logical constructs* and Ξ the *signal conversion specification*. The set of logical constructs Λ consists of three[2] sub-constructs, $\Lambda = \{\lambda_j\} = \{\lambda_s, \lambda_\ell, \lambda_p\}$, where λ_s is the *scenario bundle*, λ_ℓ is the *model of low-level logical control of human motion* and λ_p is the *procedure structure* which specify the operation elements of software embedded in artifacts. The set of logical constructs $\Lambda = \{\lambda_j\}$ receives inputs in the form of meter events $e_{met,i}(t)$, condition values $v_i(t)$, $\{v_i(t)\} \supseteq \{m_i(t)\}$, and *logistic events*, and it produces outputs in the form of control values $p_i(t)$, as logistic events $e_{lgs,j}(t)$ and *logistic values* $\{u_i(t)\}$. Logistic events and logistic values are events and values that relate to logical processing – in this case in the description of delays. They are further defined in 4.2 and 4.3. The processing performed by Λ can be written as Λ: $\{e_{met,i}(t)\}$, $\{v_i(t)\}$, $\{e_{lgs,i}(t)\} \rightarrow \{p_i(t)\}$, $\{e_{lgs,j}(t)\}$, $\{u_i(t)\}$. L sends signals conveying changes in parameters to the simulation layer Ψ. They actually take the form of *control values* $p_i(t)$. This can be written as $L : \{m_i(t)\} \rightarrow \{p_i(t)\}$, where $m_i(t) \in \mathbb{R}$ are *meter values* produced by the algorithms of the simulation layer Ψ, which receive and process control signals in order to perform the simulation and to generate meter values. Thus, the simulation layer is formally defined as $\Psi : \{p_i(t)\} \rightarrow \{m_i(t)\}$. Through evaluation of the time-dependent meter signals, the logistics layer undergoes transitions τ_i that cause changes in the values of control signals as a means to steer muscles and actuators. The time signal $t \in \{m_i(t)\}$, which is generated in the logistics layer, is a special meter signal because along with the control signals it is also transferred to the simulation.

The signal conversion specification Ξ receives inputs from the physics simulation Ψ in the form of all meter signals $\{m_i(t)\}$ and from the logical constructs Λ in the form of logistic values $\{u_i(t)\}$, logistic events $\{e_{lgs,j}(t)\}$ and logistic values $\{u_i(t)\}$. Ξ produces outputs in the form of condition values $v_i(t)$, meter events $e_{met,i}(t)$ and logistic events $\{e_{lgs,i}(t)\}$. Thus, the signal conversion specification is formally defined as Ξ: $\{m_i(t)\}$, $\{u_i(t)\}$, $\{e_{lgs,j}(t)\} \rightarrow \{v_i(t)\}$, $\{e_{met,i}(t)\}$, $\{e_{lgs,i}(t)\}$. In other words, Ξ is responsible for:

(i) converting meter signals $m_i(t)$ from the simulation layer Ψ to meter events $e_{met,i}(t)$ that are input to the logical constructs Λ. This is called *stimulus recognition*, referring to the ability of humans (and artifacts with embedded software) to recognize those stimuli from perceptive input that require action;

(ii) generating the time signal t that is used by the logical constructs Λ and the physics simulation, as well as the start event $e_{start} \in \{e_{met,i}\}$, and

(iii) regulating the *timing* of logical constructs Λ, i.e., hesitation and delays in the execution of control.

Based on these functionalities the signal conversion specification can be formally specified as $\Xi = \{\Xi_R, \xi_0, \Xi_T\}$, where

- $\Xi_R = \{\xi_{Rs}, \xi_{R\ell}, \xi_{Rp}\}$ are the specifications of stimulus recognition for human and artifact in the respective logical constructs λ_s, λ_ℓ, and λ_p;
- ξ_0 is the start specification, and
- $\Xi_T = \{\xi_{Ts}, \xi_{T\ell}, \xi_{Tp}\}$ are the specifications of human and artifact timing in the respective logical constructs λ_s, λ_ℓ, *and* λ_p.

In the next subsection we have further defined the basic processing and the constituents of the set of logical constructs and of the signal conversion specification.

3.2 Basic processing performed by the constituents of the logistics layer

Logical constructs apply control over physics simulations by changing parameters in relations. The formal specification given below involves various concepts known from finite automata representations. The involved key concepts are transitions, events, and states. The three logical sub-constructs λ_j differ in terms of the inputs they

[2] Note that it is possible to change these definitions to include more than three logical constructs. Actually, this is required if logical control in multiple humans and/or multiple distinct artifacts is to be simulated. However, this possibility has not been explored here.

receive and in the outputs they produce. As input (from λ_p and Ξ), the scenario bundle receives meter events, logistic values and events (from the procedure structure), and condition values, and as outputs (to λ_ℓ) it produces logistic events, which can formally be written as λ_s: $\{e_{met,i}(t)\}$, $\{u_i(\lambda_p,t)\}$, $\{e_{lgs,i}(\lambda_p,t)\}$, $\{v_i(t)\} \rightarrow \{e_{lgs,i}(\lambda_s,t)\}$.

The model of low-level logical control of human motion (for the rest of the article this will be shortened to *motor control model*) receives logistic values and events from λ_s together with condition values from Ξ, and it produces control signals, i.e., λ_ℓ: $\{e_{lgs,i}(,t)\}$, $\{v_i(t)\} \rightarrow \{p_i(t)\}$. As input (from λ_s and Ξ) the procedure structure receives meter events, logistic values, and events and condition values, and as output (to Ψ) it produces control signals. Thus, λ_p: $\{e_{met,i}(t)\}$, $\{u_i(\lambda_s,t)\}$, $\{e_{lgs,i}(\lambda_s,t)\}$, $\{v_i(t)\} \rightarrow \{p_i(t)\}$. Accordingly, since the distinction between high-level and low-level control that we introduced for movement control of humans has not been applied to artifacts, the processing performed by λ_s and λ_ℓ together is analogous to the processing performed by λ_p alone. Hence, λ_s and λ_ℓ together can be considered as a human interaction construct λ_h in which we can aggregate and unify all logical processing by the human. Formally, λ_h: $\{e_{met,i}(t)\}$, $\{u_i(\lambda_p,t)\}$, $\{e_{lgs,i}(\lambda_p,t)\}$, $\{v_i(t)\} \rightarrow \{p_i(t)\}$. A logical construct λ_j of the logistics layer is represented as a directed graph, whose vertices are transitions τ_i and the nodes are states Z_i, that is, $\lambda_j=\{\{\tau_i\},\{Z_i\}\}$ (Figure 4). As functions of time, τ_i and Z_i are Booleans, i.e., $\tau_i(t)$, $Z_i(t) \in \{0,1\}$ where $\tau_i(t)=1$ means 'τ_i takes place', $\tau_i(t)=0$ means 'τ_i does not take place', and $Z_i(t) = 1$ means 'Z_i is active', $Z_i(t) = 0$ means 'Z_i is not active'.

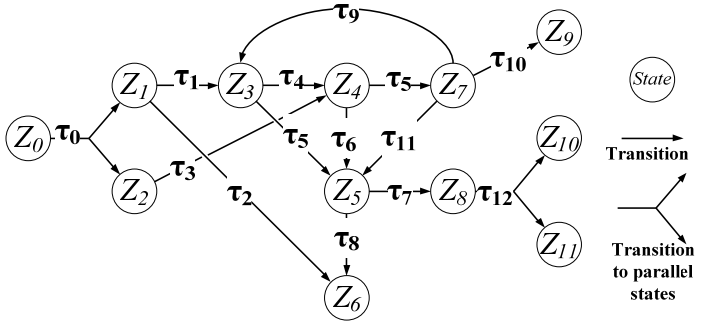


Figure 4. Example of a logical construct

Transitions between states are triggered by external events $e_{ext,i}(t)$. Each logical construct receives these from the other constructs $\{\{\lambda_j\},\Xi\}$. In order to take place, a transition may also require fulfillment of a logical condition γ_i, which can be expressed in terms of meter values, logistic values and activeness of states. Together, the values evaluated in conditions are called *condition values*[3].

Further explanation and formal elaboration on the main concepts that populate the logistics layer is given in the next section. Figure 5 gives a comprehensive overview of all the concepts introduced in Sections 3 and 4, and how they are related.

4 UNDERPINNING OF THE CONCEPTS OF DYNAMICAL PROCESSING AND ALGORITHMS OF THE LOGISTICS LAYER

4.1 Introduction

The dynamics of how the logistics layer produces control signals as output, based on meter signals as input, can be described as a propagation chain (Figure 6): a specified change in a meter value triggers a meter event, a meter event or a logistic event triggers transitions, transitions trigger changes in control variables, as well as logistic events. The propagation chain contains one loop, in which transitions trigger logistic events and these in turn trigger transitions. This loop is repeated until there are no more logistic events to be triggered. In time[4], one propagation through the whole chain of triggers, including all repetitions of the loop, has no duration. In 4.2 we

[3] Note that in 3.1 it was stated that $\{v_i(t)\} \supseteq \{m_i(t)\}$. This is true only for the condition values $v_i(t)$ mentioned in that particular statement, which are output of the signal conversion specification.

[4] Throughout this article, *time* refers to time in the simulated use process, not to computation time.

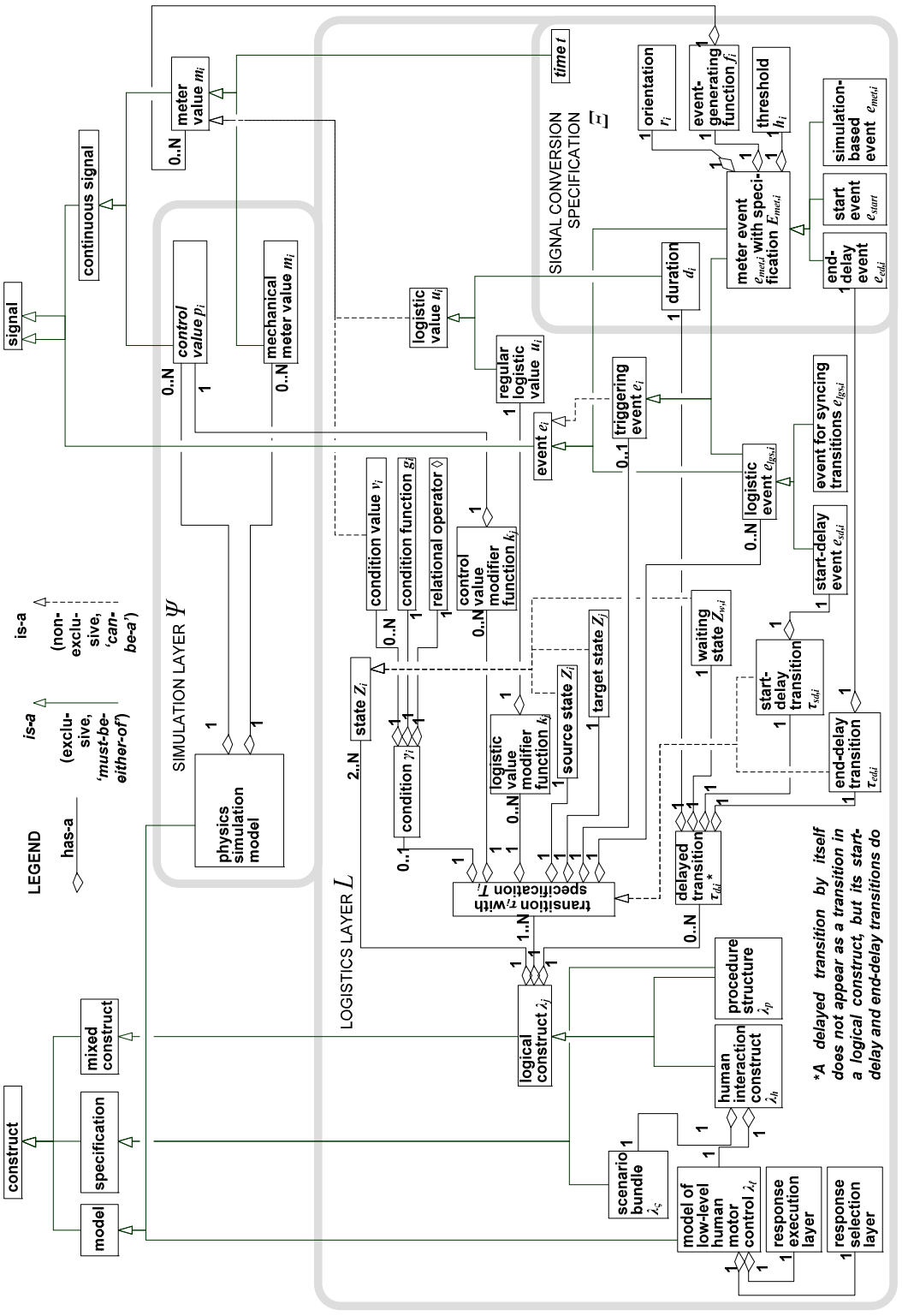


Figure 5. Entity-relationships diagram of the concepts populating the simulation layer and the logistics layer

will formally specify and interpret these concepts in their order of appearance in the propagation chain. To be able to specify *delays* in information processing by humans and artifacts, some additional definitions are needed, which are presented in 4.3.

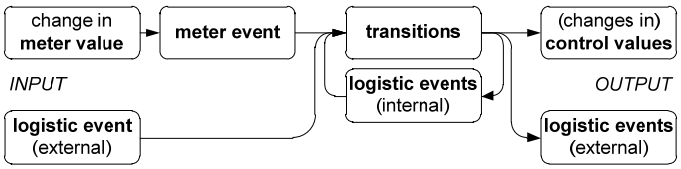


Figure 6. Propagation chain of the logistics layer

4.2 Definition of dynamical concepts in the logistics layer

Meter variables are specified in the simulation layer Ψ as output signals, which are to be generated after each simulation time increment Δt_{sim}. Transfer of meter values $m_i(t)$ to the logistics layer L takes place with an increment, or communication interval, Δt_{com} , with $\Delta t_{com} = n_{sample} \cdot \Delta t_{sim}$, $n_{sample} \geq 1$. In each case when the time in the logistics layer is updated, $t := t + \Delta t_{com}$, new meter values $m_i(t)$ are read. The *initial values* of meter variables $m_i(t_0)$ typically describe a static state of the human-artifact system.

In general, an *event* can be defined as the occurrence of a particular measurable change in a process. The change is specified because it is considered to be a trigger of transitions in the logistics layer. Since it occurs at a point in time, an event $e_i(t)$ has no duration. An event is either internal or external, and it is either a logistic event or a meter event. An internal event reflects changes inside a given logical construct, while an external event reflects changes in another logical construct or in the simulation layer.

Logistic events correspond to specified changes in a logical construct. They express a change that occurs because (i) a particular transition has taken place, or (ii) a particular state has become active or inactive. As can be seen in Figure 6, a logistic event is always a consequence of a meter event occurring at the same time. Logistic events allow for, respectively, (i) synchronization of transitions that must take place at the same time, and (ii) a shorter description of the occurrence of any transition into, or out of, a given state. A logistic event $e_{lgs,i}$ (λ_j,t) can be external or internal. *Meter events* are external events corresponding to specified changes in meter variables, i.e., in the simulation outputs or time. How meter events $e_{met,i}$ (t) (which are discontinuous in time) are generated from meter variables $m_i(t)$ (which are continuous functions of time) is defined as follows. Let $\{m_i(t)\}$ be the set of meter variables and let us define a subset of meter variables $\{\mu_i(t)\} \supseteq \{m_i(t)\}$ as *recognition variables,* which are used to specify meter events $e_{met,i}$ (t). Now we can define the *meter event specification*, $E_{met,i} = \{e_{met,i}\ (t), r_i, h_i, f_i(t)\}$, where $e_{met,i}$ (t) is a Boolean: $e_{met,i}(t) \in \{0,1\}$, $r_i \in \{\uparrow,\downarrow,\updownarrow\}$ is the *orientation* of the event: increasing, decreasing or bidirectional, $h_i \in \mathbb{R}$ is the *event threshold,* and $f_i(\mu_1(t),\mu_2(t),..,\mu_n(t))$: $\mathbb{R}^n \rightarrow \mathbb{R}$ is the *event-generating function*. In the simplest case, the event-generating function reflects changes in the value of a single recognition variable, e.g., $f_i = \mu_1(t)$ (where μ_1 could be for instance the distance between the human fingertip and a button that must be pushed), but it can also be an expression that requires evaluation of multiple recognition variables, e.g., $f_i = 5\mu_1(t) + \sqrt{}(\mu_2(t)) - {}^{d}/_{dt}(\mu_3(t))$. The meter event occurs if the value of the function $f_i(t)$ crosses the threshold in the direction(s) specified by the orientation (Figure 7).

This is formally defined as:

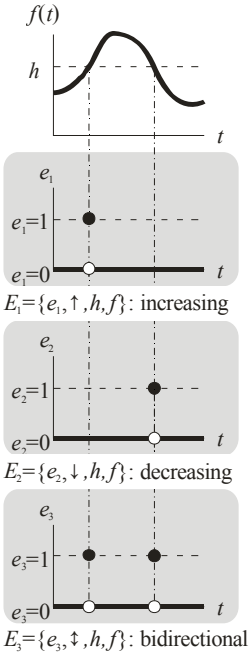


Figure 7. Occurrences of meter events with different orientations

$$e_{met,i}(t) = 1 \text{ iff } f_i(t) = h_i \wedge \left(\left(r_i = \uparrow \wedge \frac{df}{dt} > 0 \right) \vee \left(r_i = \downarrow \wedge \frac{df}{dt} < 0 \right) \vee \left(r_i = \updownarrow \wedge \frac{df}{dt} \neq 0 \right) \right)$$

The start event e_{start} with specification $E_{start} = \{\uparrow, t_0, t\}$ starts the simulation at $t = t_0$. It is part of the start specification ξ_0, which prescribes that when the system user enters a start command, the start event is sent to the scenario bundle and to the procedure structure. At the same time, the time signal is set to $t = t_0$. The scenario bundle and the procedure structure must contain an initial transition τ_0 that takes place when e_{start} occurs. Before the start of the simulation, all meter events are zero: $t < t_0 : \forall i : e_{met,i}(t) = 0$.

A *transition,* τ_i, with condition γ_i connects a *source state,* Z_i, to one or more *target states* $Z_j(\tau_i), j = 1,..,n_{ts}$, where $n_{ts} \geq 1$. If $n_{ts} > 1$, then $Z_j(\tau_i)$ are called parallel target states of τ_i. The transition specification T_i of a transition τ_i can be written as $T_i = \{Z_i, Z_j, e_i, \gamma_i(v_1(t), v_2(t),...), \{k_i\}\}$, with

- $\gamma_i = \{g_i(v_1(t), v_2(t),...), \Diamond\} \in \{0,1\}$ the *transition condition*, where $g_i: \mathbb{R}^n \rightarrow \mathbb{R}$ is a function of specified condition values $v_i(t)$ $\{v_i(t)\} \supseteq \{m_i(t)\} \cup \{u_i(t)\} \cup \{Z_i(t)\}$, $\Diamond \in \{ =, \neq, <, >, \leq, \geq \}$ is a relational operator, so that $\gamma_i = 1$ if $g_i \Diamond 0$, and
- $k_i(p_i(t), m_i(t))$, $k_i: \mathbb{R}^n \rightarrow \mathbb{R}$ a set of modifier functions that, if τ_i takes place, change control variables p_i and logistic values u_i as follows: $p_1 := k_1(t)$, …, $p_P := k_P(t)$, $u_1 := k_{P+1}(t)$, $u_U := k_{P+U}(t)$,where P is the total number of control variables and U the number of logistic values that have been specified to be modified by τ. The control variables and logistic are thus specified as functions of values of other control variables and logistic variables as well as meter variables: $k_j(t) = k_j(p_1(t), p_2(t),..., u_1(t), u_2(t),..., m_1(t), m_2(t),...)$. The newly assigned functions k_j keep controlling these values until another transition takes place that specifies a different set of modifier functions.

A transition τ_i with transition specification T_i takes place at a time t, i.e., $\tau_i(t) = 1$ iff $e_i(t) \wedge \gamma_i(v_1(t), v_2(t),\ldots) \wedge \varepsilon(\tau_i, t)$, with $\varepsilon(\tau_i,t) \in \{0,1\}$ the *enabling*, meaning that the transition is enabled, $\varepsilon(\tau_i,t) = 1$, when $Z_i(t) = 1$, i.e., its source state is active. Since events have no duration, transitions do not have a duration either. As soon as a transition takes place, its source state Z_i becomes inactive and its target states $Z_j(\tau_i)$ become active. Both the scenario bundle and the procedure structure must contain an initial transition $\tau_0(e_{start})$ that takes place when e_{start} occurs.

A *control variable* is a means of manipulating a relation in the physics simulation model Ψ. It allows us to change the value of the relation, which is then considered at computing the conduct of physics processes. Typical control variables are, for instance, prescribed speeds or prescribed forces to be exerted by actuators. Certain variables in relations, such as mass and Young's modulus cannot appear as control variables.

4.3 Definition of delays in logical processing

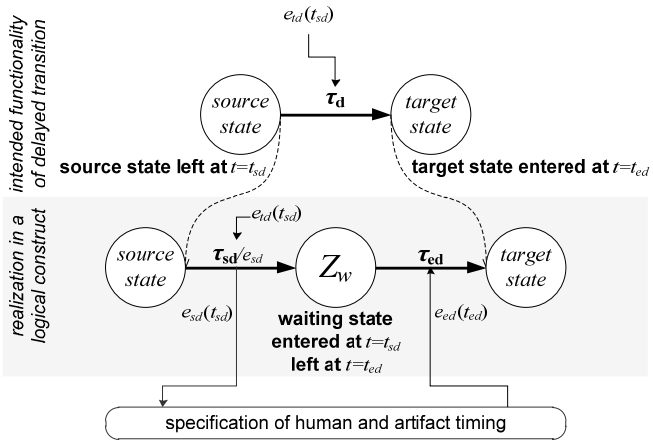
The option of specifying delayed transitions can be useful to investigate the effects of latency in artifactual or human information processing[5]. A delay in artifactual information processing can for instance be a deliberate time-out to wait for a response from the user, or an unavoidable delay caused by data exchange with a central server. A delay in human information processing can for instance be hesitation to proceed with an action. If during such hesitation, operation of the product might proceed so that different hesitation durations might lead to different unfoldings of the use process. The designer can perform what-if studies with various explicitly entered durations of

[5] This section does not deal with delays in physical processes related to phenomena such as hysteresis, damping and inertia. Such delays have to be dealt with in the simulation model.

hesitations - for instance to fine-tune a timeout in the product's programming.

In terms of specifying the scenario bundle and the procedure structure, such a delay means that the logistic events and changes in control values resulting from the transition do not take place at the same time as its triggering event, but some time later. Formally, a delayed transition $\tau_{d,i}$ in a logical construct λ_j is triggered at $t = t_{sd,i}$ by an event $e_{td,i}$ (*delay-triggering event*) but it takes place (i.e., λ_j assumes its target state) at $t = t_{sd,i} + d_i$, where d_i is the delay in seconds. This cannot be realized directly with a transition according to the definitions in the previous section. However, as a workaround it can be realized by specifying a delayed transition as a compound of two regular transitions $\tau_{sd,i}$, $\tau_{ed,i}$ with a waiting state $Z_{w,i}$ in between: $\tau_{d,i} = \{\tau_{sd,i}, Z_{w,i}, \tau_{ed,i}\}$.

The start-delay transition $\tau_{sd,i}$ is in fact a dummy transition, while the end-delay transition $\tau_{ed,i}$ is the actual delayed transition (Figure 8). When at $t = t_{sd,i}$ a meter event takes place that triggers the start-delay transition $\tau_{sd,i}$, the logical construct assumes the waiting state $Z_{w,i}$. The start-delay transition $\tau_{sd,i}$ generates a logistic event called *start-delay event*, $e_{sd,i}$. Because processing one propagation chain (Figure 6) in a logical construct λ_j has no duration, the delay is 'put aside' outside λ_j, to be processed in a next propagation chain at $t = t_{sd,i} + d_i$. For this 'putting aside', we have included the specifications of human and artifact timing $\Xi_T = \{\xi_{Ts}, \xi_{T\ell}, \xi_{Tp}\}$. After ξ_{Tj} ($j = \{s,\ell,p\}$) has received the start-delay event it generates the *end-delay event*, $e_{ed,i}$ at $t = t_{ed,i} = t_{sd,i} + d_i$ which is sent back to λ_j where it triggers the *end-delay transition* $\tau_{ed,i}$ out of the waiting state $Z_{w,i}$. As a whole, a specification of human or artifact timing can be defined as $\xi_{Tj} = \{\{e_{sd,j}\},\{d_j\},\{e_{ed,j}\}\}$. In this specification, $\{e_{sd,j}\} \supseteq \{e_{lgs,j}\}$ is the set of start-delay events to be received from the logical construct λ_j, while $\{d_j\}$ is the set of durations of the delays, and $\{e_{ed,j}\}$ is the set of end-delay events to be sent to λ_j.



(The notation τ_{sd}/e_{sd} is a common convention to specify "the transition τ_{sd} generates the logistic event e_{sd}")

Figure 8. Realization of a delayed transition in a logical construct.

4.4 Concluding remarks

In Sections 3 and 4 we have purposefully derived the theoretical elements that are needed to achieve the targeted system functionality. The theory explains the principles of the logistics layer containing the scenario bundle, which is a logical construct acting as a control mechanism over the simulation. This makes it possible to simulate procedurally disjunct sequences of interactions based on the designer's conjecture of human decision-making. To connect simulation models and scenario bundles, and to include information processing by artifacts in simulations, we have defined two additional logical constructs and a signal conversion specification.

The theory does not extend to the realization of software components for storing and processing constructs, models, and specifications. We have addressed these implementation issues in the next section. Section 5 also addresses the arrangement of, and data exchange between the software components in a way that unifies the processing scheme in Figure 3 and the reasoning model of human-artifact interaction in Figure 2.

5 PROTOTYPE IMPLEMENTATION

5.1 Objectives

As a proof-of-the concept implementation, we have realized a working prototype of the hypothesized new functionality based on the theory. The objectives were (i) to gain experiences with feasibility and computability in general and (ii) to demonstrate the functionality of the conceptualized system. The objective of feasibility testing was to confirm that the theory can be converted into a structured set of processable algorithms and to demonstrate the usability. We have further elaborated it in Section 6 of this article. We have addressed the other objectives in other work, as is briefly discussed in Section 7.

The next subsection, 5.2, is a specification and explanation of how the theoretical resources presented in Sections 3 and 4 are converted into the functional processing units that we have later on realized in computer software. It has to be mentioned that in 5.2 we have kept detailing of the units and the connections independent of implementation tools (i.e., programming language, etc.). Subsequently, in 5.3, we will elaborate the approach for the proof-of-the concept implementation and, based on functional requirements, selected the actual software tools. In 5.4 and 5.5, we will describe the activities with the selected software to create control specifications and to connect them to simulation tools.

Table 1. Principal constructs of resource-integrated modeling and simulation

layer / *source of instantiation*	*logistic layer (execution of logical constructs)*		*simulation layer (simulation of physics)*
designer's conjecture	scenario bundle	signal conversion specification	
designer's creational skills	procedure structure		simulation model
predefined	motor control model		

5.2 Conversion of theoretical resources to the requested functionality

5.2.1 Operationalization of the conceptual model

We have decomposed the conceptual model for controlled simulations presented in Figure 3 into five principal constructs as we have theoretically defined in Sections 3 and 4. From an implementational viewpoint, these constructs can be distinguished based on (i) their source of instantiation (i.e., whether they are instantiated by the designer based on conjecture or based on creational skills, or predefined) and (ii) the layer they reside in (i.e., the logistic layer or the physics simulation layer) as shown in Table 2.

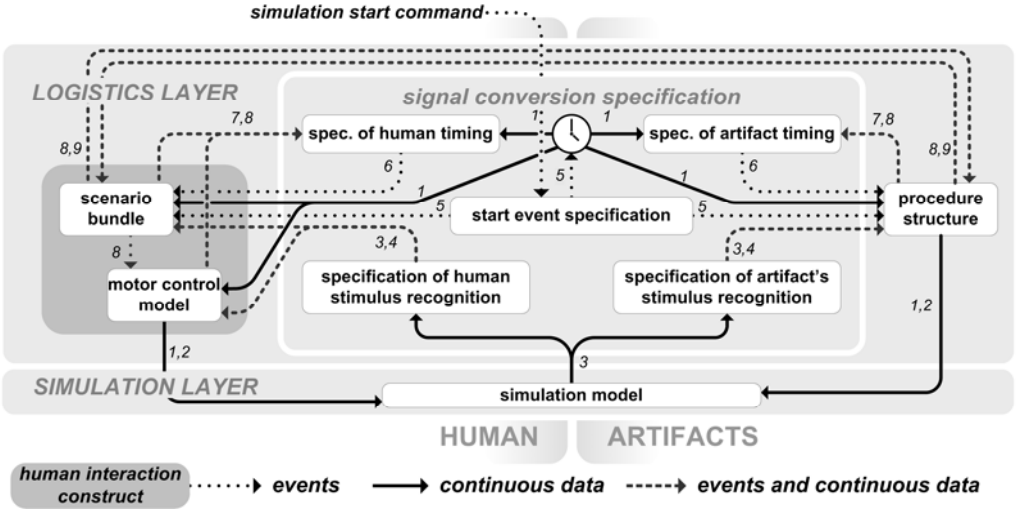


Figure 9. Signal flows of controlled interaction simulation

Table 2. Specification of the labels used in Figure 9.

1. time $t \in \{m_i\}$	6. end-delay events $\{e_{ed,i}\} \supseteq \{e_{met,i}\}$
2. control signals $\{p_i\}$	7. start-delay events $\{e_{sd,i}\} \supseteq \{e_{lgs,i}\}$
3. meter signals $\{m_i\}$	8. logistic events $\{e_{lgs,i}\}$
4. meter events $\{e_{met,i}\}$	9. logistic values $\{u_i\}$
5. start event $e_{start} \in \{e_{met,i}\}$	

Figure 9 shows the general signal flows of controlled interaction simulation as a result of combining the theory in Sections 3 and 4 with its operationalization in 5.2.1. This figure maps the processing flows in Figure 3 to the hypothesized reasoning model of human-product interaction in Figure 2. The labels are explained in Table 1.

5.2.2 Logical constructs: scenario bundle, motor control model and procedure structure

A *scenario bundle* λ_s is a logical construct by which designers specify interactions, operations, and behaviors in conceived use processes representing the related human decision-making. From a computational point of view, a scenario is a compilation of transitions τ_i corresponding to *decisions* that the user of the product is supposed to make in order to start or end particular *activities* specified as states Z_i.

The instructions specified in the scenario bundle correspond to human control that manifests itself on the level of decision making. They specify high-level activities such as, for example, *'push button', 'insert coin into slot', 'turn left',* or *'open lid'* in conceived use processes. The activities are represented as states and decisions are represented as transitions between states (e.g., a decision to change from state *'do nothing'* to state *'pull lever'*). The instructions attached to transitions and states do not specify the body parts or muscles to be addressed, and specify values of control signals only qualitatively (e.g., 'raise forearm fast'). The advantage of scenario bundle-based specification of interactions is that the designer does not have to put efforts in specifying low-level control.

In terms of signal flows this means that the scenario bundle sends outputs to the motor control model rather than generating output signals to the physics simulation model directly. The mentioned communication uses logistic events, which signify the points in time when high-level activities start or end. The activities themselves correspond to states Z_i in the scenario bundle, which carry the names of the activities. One high-level activity Z_i in the scenario bundle, may correspond to a group of low-level activities $\{Z_i'\}$ in the motor control model. The waiting states $Z_{w,i}$ that were introduced in 4.3 also correspond to 'activities', but by way of exception, logistic events evoked by transitions to and from waiting states are sent to the signal conversion specification rather than to the motor control model.

As explained above, the *motor control model* λ_ℓ is a logical construct describing motion control of specific body parts with the purpose of generating control signals for muscles in the physics simulation model. For interaction with a specific product, a motor control model is instantiated based on various chunks of knowledge from human motor science about human response selection and response execution. It is assumed that many activities in the scenario bundle can be recognized as common interaction patterns and that their variations can be parameterized. For these activities, the process of instantiation can be automated, e.g., 'reaching with the hand for a point (x, y, z)', or 'pushing a button with stroke s and orientation (α, β, γ)'.

Together the motor control model and the scenario bundle

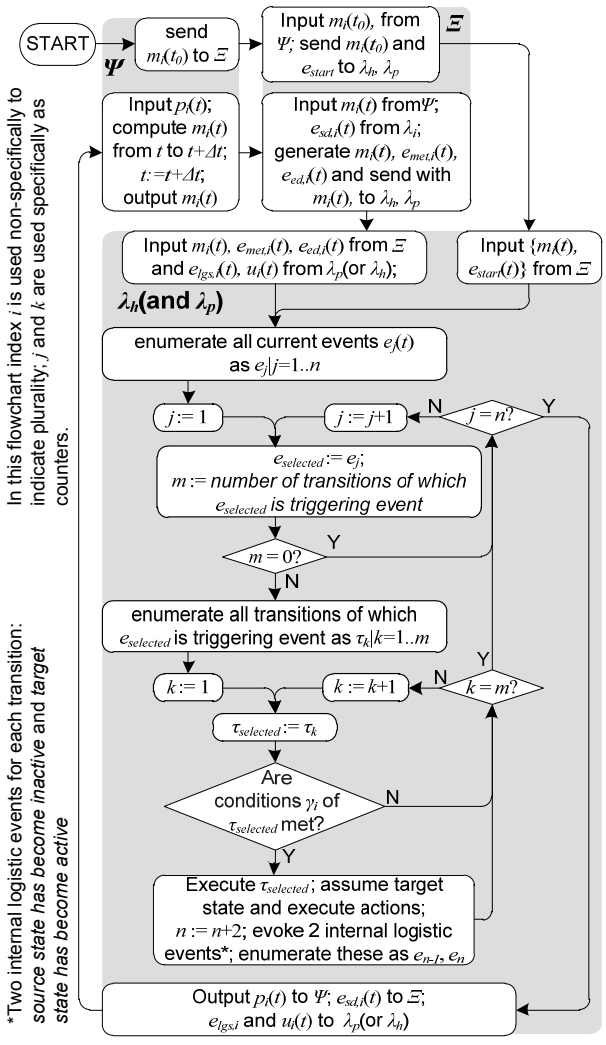


Figure 10 Processing flowchart of controlled simulation

are processed as a human interaction construct λ_h. The optional artifactual counterpart of λ_h is the *procedure structure,* λ_p, that makes it possible to include the artifactual counterparts of decision-making, and muscle activation. It structure represents the instructions programmed into an artifact's control mechanisms according to the intents of designers and as such it can be seen as a bundle of scenarios from a product perspective.

5.2.3 Signal conversion specification and general process flows of controlled interaction simulation

Figure 9 also shows how the signal conversion specification is decomposed into functional sub-specifications and which signals they exchange. The conversions in the signal conversion specification can be considered as applying predefined 'templates' of operations to signals specified in logical constructs and in the physics simulation model. This means that the contents of the signal conversion specification can be automatically extracted and configured without explicit involvement of the designer.

In Figure 10, the order of performing processing steps in time during controlled simulation is shown as a flowchart, focusing on processing of events and transitions by the logical constructs. When specifying processing steps, we have to deal with the practical implication that simultaneous events and transitions must put into a processing order before their effects can be computed. To that end, the flowchart in Figure 10 refers to enumeration of events and transitions. We have not elaborated detailed processing steps for enumeration. In the ultimate processing algorithm an arbitrary enumeration scheme can be applied as long as it is consistent and unambiguous (e.g., from top to bottom and then from left to right as items graphically appear in λ_j), and available to system users. It has to be mentioned that only in exceptional cases the exact processing order is known to have influence on the output of a finite automaton [36]. We have not included an 'end simulation' command in the flowchart. The controlled simulation may be ended at any time by stopping the computational processes.

5.3 Selection of tools for a proof-of-concept implementation

5.3.1 Functional requirements and realization approach

To transfer the theory to a fully functional software solution that allows multi-aspect investigation of use processes, we reasoned that software components with the following functionalities are needed:

(i) A model repository describing anatomy, geometry, and mechanical properties of a variety of human bodies;
(ii) An artifact modeling system, or a system that supports the conversion from conventional CAD models and that allows insertion of models from (i), that supports inclusion of mechanical properties and specification of meter values;
(iii) A repository of motor control models tailored for controlling the muscles in the models (i);
(iv) A subsystem for the specification of logical constructs that allows insertion of models (iii);
(v) A subsystem that generates a signal conversion specification, and connects input/output signals of the simulation model and logical constructs;
(vi) A subsystem for mechanics simulation of the models created with (or produced by) (ii);
(vii) A subsystem for execution of the logical constructs specified with (iv).
(viii) A subsystem for concurrent processing of (vi) and (vii), providing output to designers in the form of animated motion of the simulation model and numerical values of variables over time, as well as a visualization of the 'path' taken through the scenario bundle.

The proposed theory underpins the functionalities realized by components (iv), (v), (vii), and (viii). According to our best knowledge, no software tool or package that is able to provide this combined functionality is available. Consequently, we had to develop and/or adapt the resources to make the proof-of-concept implementation possible. In their functioning, these components depend on all the other components: (iv) needs (iii), (v) and (viii) need (vi),

which in turn needs (i) and (ii). Forerunning surveys pointed out that no commercial software is available to realize these other functionalities [37-39]. Nevertheless, we had to implement and adapt these into the proof of concept in some way. Since our primary goal has been to prove the feasibility of the functionalities of (iv), (v), (vii), and (viii), we have decided to realize the proof-of-concept prototype with existing adaptable tools. Using such tools we have realized (i) specification of logical constructs, (ii), generation of the signal conversion specification, (iii) execution of logical constructs, and (iv) concurrent processing during simulations, to create simulatable/executable models and constructs according to theory. As we have documented below, some minor deviations to the theoretical concepts had to be introduced.

Since the theory does not address the user interface for the creation of constructs, we have used the interfaces offered by the selected tools (which we will specify in 5.3.4 and 5.3.5.). We have realized the human body model and the motor control model as *proxies* by using existing systems. For these models and their simulation, the priority has been that they exchange signals with the logistics layer as we have specified in the theory.

5.3.2 Human body modeling

Several of the commercially available software packages mentioned in the review in Section 2 deploy readily prepared human body models in mechanical interaction simulations with artifacts. Using a comprehensive human body model offered by one of these packages would compel us to use the dedicated mechanics simulation capabilities offered by that same package. However, since we wanted to include a conceptual solution for large-deformation simulation of flexible-tissues, we needed the flexibility and configurability offered by a general-purpose mechanics simulation package. Therefore we decided to build only a simplified partial model of the human body that allowed us to test some typical interactions such as grasping and pushing. We have elaborated the conceptual solution for large-deformation simulation in our papers on *nucleus-based modeling* [4, 40]. For the work described here, the reader does not need to be familiar with nucleus-based modeling.

As we did not use a full-human-body simulation package, we could not implement system functionality to instantiate varying human bodies from a repository. Since one (partial) instance of a human body model suffices for the goals described in 5.1, and since the feasibility of this functionality has already been proven by the existing packages, we deemed it unnecessary to include and utilize it.

5.3.3 Modeling for mechanical simulation

The two groups of commercialized systems for modeling and simulation of mechanical behavior commonly used by the industry are finite-element method systems (FEM) and multibody dynamics systems (MBS) [41]. Essentially, the capabilities of these two types of systems are complementary [37]. FEM systems allow simulations of deformations in continua (such as flexible human body tissues) but they are not very suitable for simulation of the kinematics and kinetics of multiple interacting objects, which is the domain of MBS approaches. The latter, on the other hand, generally lack support for simulating deformations. The most common workarounds to simulate both multibody kinetics and deformations are (i) co-simulation between FEM and MBS, and (ii) discrete-flexibility modeling within MBS.

Co-simulation is often based on open-loop coupling [e.g., 42], by which computed deformation of the FEM model is not reflected as a change of geometry in the MBS model. Due to computational complexity, the application of closed-loop co-simulations to 3D flexible bodies has been scarce [43]. To allow closing of the loop, FEM models are typically kept one-dimensional (i.e., rods and beams). On the other hand, discrete flexibility [44] enables simulations of large deformations by applying discretization into ‘particles’ in conventional MBS software, and connecting these by spring-dampers. Actually, in our nucleus-based modeling and simulation approach (see 5.3.2) flexible bodies are decomposed the same way. To benefit from this correspondence, we decided to equip the proof-

of-concept system with solely an MBS system as a proxy for simulation of all mechanical behavior, including deformations.

At selecting a commercial MBS system for our purpose, the following practical and implementation-related requirements apply:

(i) There must be a possibility to control simulations with values obtained from logic processing;
(ii) It should be possible to define behavioral models using imported 3D CAD models, and to edit and manipulate models in a 3D environment;
(iii) It should be possible to visualize simulation results by animated motion of 3D human/artifact models.

Many of the packages available on the market fulfill these requirements, for instance, LMS VirtualLab, Simpack, and MSC Adams [45], Working Model 2D, and VisualNastran 4D [46]. In fact, in the various stages of developing our proof of concept we have worked with all the three last-mentioned systems, using Adams for the latest implementation[6].

5.3.4 Specification and execution of logical constructs

Since by their definitions the logical constructs introduced in 3.2 are equivalent to existing representations of finite automata (FA) we decided to use commercially available FA software as a proxy to specify and execute logical constructs. For our purposes the selected software had to fulfill the following requirements:

(i) The software must allow its user to maintain distinct constructs for (a) the scenario bundle, (b) the models of low-level logical human control, and (c) the procedure structure. The software should enable control of a single simulation model through concurrent execution of these constructs.
(ii) Based on the definitions in Sections 3 and 4, the software must allow (a) interfacing with simulations to exchange meter and control signals, and (b) specification of events, transitions, states, and transition conditions.
(iii) The software must support specification and modification of logical constructs by a graphical representation
(iv) It should be possible to monitor (or review) the succession of transitions and states during (or after) a simulation

Requirement (i) means that the graphical representation that is used should have the representation potential of supporting concurrency and hierarchy. Since they offer the required representation potential and also appear to be the most widely used FA representations (see Section 2), we narrowed down our search to statecharts and Petri nets. Requirement (ii) addresses interfacing with the physics simulated by an MBS system. These systems are usually[7] capable of exchanging control-related signals with Matlab Simulink, which offers a Stateflow toolbox that enables specification of FA using a statechart dialect [47]. This toolbox is widely used in the industrial practice [48]. It fulfills the above requirements with two restrictions, namely, that (ii) meter events must be specified in Simulink outside Stateflow, and (iv) successions of transitions and states can be monitored during simulations, but not afterwards. Considering this, we have selected Simulink Stateflow for specification and execution of logical constructs. An additional advantage is that by using Stateflow in prototyping of the logistics layer, we could realize the proof-of-concept prototype by using only two software packages (Simulink and an MBS system).

5.3.5 Motor control models of human motion

In the pilot implementation, we have used proxy models of human motor control rather than validated models. At the time of starting our project, no validated models were known to be available and we did not consider it feasible to develop them with the available resources. However, in the meantime new developments have indicated that,

[6] This package has the additional advantage of allowing integrated simulations with the human-body package LifeModeler, which has been developed as a spin-off of Adams. However, as argued in 5.3.3, we did not exploit that opportunity.
[7] We verified this for Adams, VirtualLab, Simpack , and VisualNastran 4D.

with some limitations, simulation of motor control based on validated models is possible (see Section 2, references [8, 13, 15]).

The proxy construct of low-level control of human motions bridges the gap between the scenario bundle and the simulation. It has to convert the logistic events from the scenario bundle to quantitative control values for the simulation. For the proxy we adopted an approach often taken in programming robotic human-body imitations, where programmed instruction are typically logic-based, and aimed at obtaining successful interaction *results* rather than at realistic motions *during* the interaction[8]. In the same manner we have 'programmed' instructions, for instance for moving the fingertip from A to B, without bothering whether the used motion patterns were natural and efficient. For a future implementation, replacement of these workarounds by validated models is to be considered.

5.4 Specification of control instructions with Simulink Stateflow

5.4.1 Implementation of basic specification elements

The further elaboration of logical constructs is organized as follows. First, in this subsection we have elaborated the proof-of-concept implementation of the basic specification elements by mapping Stateflow elements to the elements defined in Section 4. After that, specific implementation aspects of the three different logical constructs and the signal conversion specification are elaborated in 5.4.2.

The Stateflow dialect [47] is different from the original statecharts [28] in two respects: (i) it offers various enhancements in the form of building blocks for logical processing, and (ii) the graphical representation of parallel states is slightly different. The two Stateflow enhancements we used in the proof-of-concept implementation are (i) implicit logistic events, (ii) connective junctions and (iii) subcharting. Implicit logistic events reduce the number of events that the system user has to specify. Connective junctions offer a convenient way to graphically combine transitions that are triggered by the same event and/or share part of their conditions or effectuated control commands (*actions*, as explained below). To facilitate hierarchical decomposition, *subcharting* allows demoting connected groups of states and transitions to subcharts, the contents of which are hidden from higher-level views. The appearance of the graphical elements in Stateflow is illustrated with our application example in Section 6.

In accordance with the definitions in Section 4, we have implemented two basic specification elements using Stateflow charts: (i) states and transitions with transition conditions, and (ii) logistic events. This has been done as follows. Just like in the logical construct in Figure 4, states and transitions in Stateflow charts are specified as nodes and vertices. As will be illustrated in 6.2, parallel states are specified as child states of a parent state, which is different from the arrangement in Figure 4. To each state Z_i, a name is assigned. After the name, *actions* can be specified to be executed either with any incoming transition (specified as `entry:action` or `during:action`) or with any outgoing transition (specified as `exit:action`). In logical constructs, actions are used to generate logistic events and to apply modifier functions k_i to control and logistics variables. Modifier functions (see 4.2) can be assigned as `entry: p_1=k_1, p_2=k_2`, etc., or `exit: p_1=k_1, p_2=k_2`, etc. if they are step functions, i.e., they assign a new constant value to a variable. To assign regular functions, they have to be specified as `during: p_1=function_expression_1, p_2=function_expression_2`, etc. to ensure that the function expression is continuously evaluated till the next transition.

A transition can have a triggering event $e_i(t)$ assigned to it. This can be a meter event imported from the signal conversion specification or an internal or external logistic event. If no event is specified, any event occurring when the transition is enabled will trigger it. Actions to generate logistic events and/or to apply control modifiers as step functions to control variables can also be assigned to transitions. This is specified as `/action`. Transition conditions

[8] For instance, for the Utah/MIT dextrous hand [49] it was more important to obtain grip on an object so that it could be carried, than to achieve realism in the preceding motions of the fingers and the arm.

$\gamma_i = g_i((v_1(t),v_2(t),...)) \lozenge 0$, are specified as `[g_i ◊ 0]`, e.g., `[x + 3*y - 4 > 0]`. Modifier functions for stepwise changes can be assigned to τ in the following way: `/p 1=k 1, p_2=k_2`, etc.

A logistic event $e_{lgs,i}(t)$ is used to synchronize transitions within one logical construct or across multiple logical constructs. Stateflow distinguishes two types of logistic events: explicit logistic events and implicit logistic events. Explicit events are specified in the Stateflow chart as actions. This is done either by assigning an action `/expl event` to a transition τ_i or by assigning an event `entry:expl event` or `exit:expl event` to a state Z_i. An implicit logistic event is an event that is automatically generated and named by Stateflow when a state becomes active or inactive. The implicit logistic event is called `enter[state name]` or `exit[state name]`, respectively. Table 3 shows how explicit and implicit events are used to synchronize transitions τ_i.

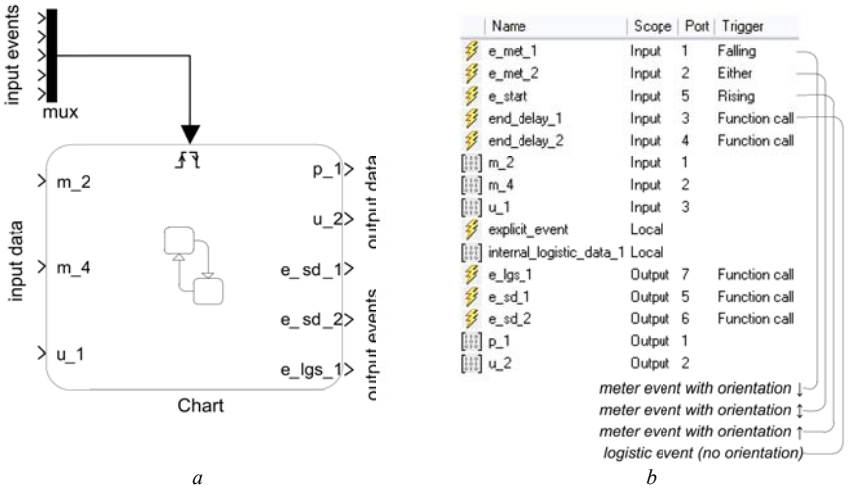


Figure 11. *a:* Example of a Stateflow 'Chart' block with external connections in Matlab Simulink. *b:* specification in the Simulink model explorer

Table 3 Synchronization of transitions through explicit and implicit logistic events.

EXPLICIT EVENTS	IF the action	`/expl_event`	is assigned to a transition τ_i	and `expl event` is the triggering event of τ_j, THEN τ_j takes place synchronously with	τ_i
		`entry:expl event` or `en: expl_event`	is assigned to a state z_i		any transition of which z_i is the target state
		`exit:expl_event` or `ex:expl_event`			any transition of which z_i is the source state
IMPLICIT EVENTS	IF	`enter[z_i]` or `en[z_i]`	is the triggering event of τ_j, THEN τ_i, takes place synchronously with		any transition of which z_i is the target state
		`exit[z i]` or `ex[z_i]`			any transition of which z_i is the source state

Based on these elementary building blocks of logical constructs, the implementation of the constructs as modeling and specification elements in their entirety is elaborated in the next subsection.

5.4.2 Implementation of the logical constructs λ_s, λ_ℓ, λ_p, and the signal conversion specification Ξ

Logical constructs λ_j are instantiated as Stateflow charts in Matlab Simulink block diagrams and are connected with other blocks through data ports and event ports as shown in Figure 11a. They have 'internal' data signals and events, which are not exchanged with other Simulink blocks, while the external data and event ports, as well as internal data and event signals are specified and named using the Simulink model explorer (Figure 11*b*).

In the proof-of-concept implementation we could specify the three constructs by three separate Stateflow charts as described above. However, since the constructs λ_s and λ_ℓ, which control human motions, are closely interlinked through logistic events sent from λ_s to λ_ℓ, we have combined the two into one human interaction construct, λ_h (see also 3.2). This way, we could simply specify synchronization of transitions in the two constructs by implicit events. Also, we could use explicit events 'internally' throughout both constructs.

In λ_h, we have further decomposed the motor control model into two layers corresponding to the two levels distinguished by Stelmach [49], i.e., response selection and response execution (for further explanation, see next paragraph and [39]). Thus, by considering the scenario bundle as corresponding to the decision-making level above these two, λ_h can be considered as structured into three layers: the scenario layer, the response selection layer, and the response execution layer (Figure 13).

The behavior descriptions in the response selection layer appear as subcharts of states in the scenario bundle (e.g., 'retract upper arm' as one of the substates appearing in the subchart of 'pull lever'). To execute motion patterns, the behavior descriptions in the response-execution layer specify control-variable modifier functions. These basic low-level control commands for the movement of each limb for each one of its degrees of freedom are represented by *response-execution primitives* (Figure 13): e.g., 'move forward', 'rest', and 'move backward', with specified changes in angular velocity.

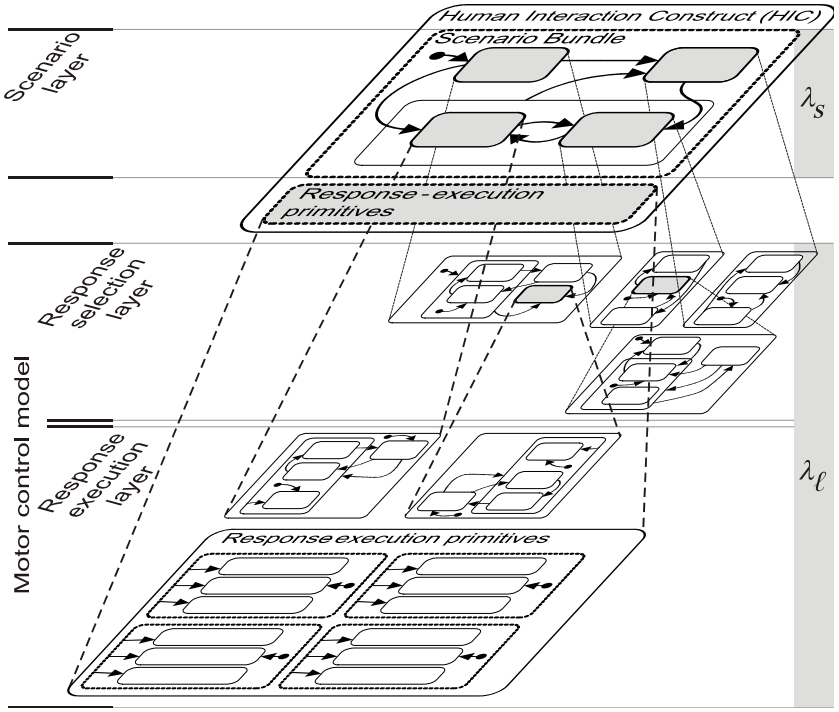


Figure 13. Layered structuring of the human interaction construct λ_h

Figure 12a shows an example of a signal conversion specification the proof-of-concept implementation in accordance with Figure 9 in 5.2.3. Figure 12b shows how specifications for stimulus recognition, which convert meter signals to meter events, are instantiated. Meter signals that have been selected as recognition signals are first processed by an event-generating function, i.e., a Simulink subsystem `Event_gener_func_i`, that can represent any purposeful operation performed on one or more variables. Its result is led through a hit-crossing block that is specified by parameters for the orientation and threshold value of the event. Without event-generating function, the *specification*

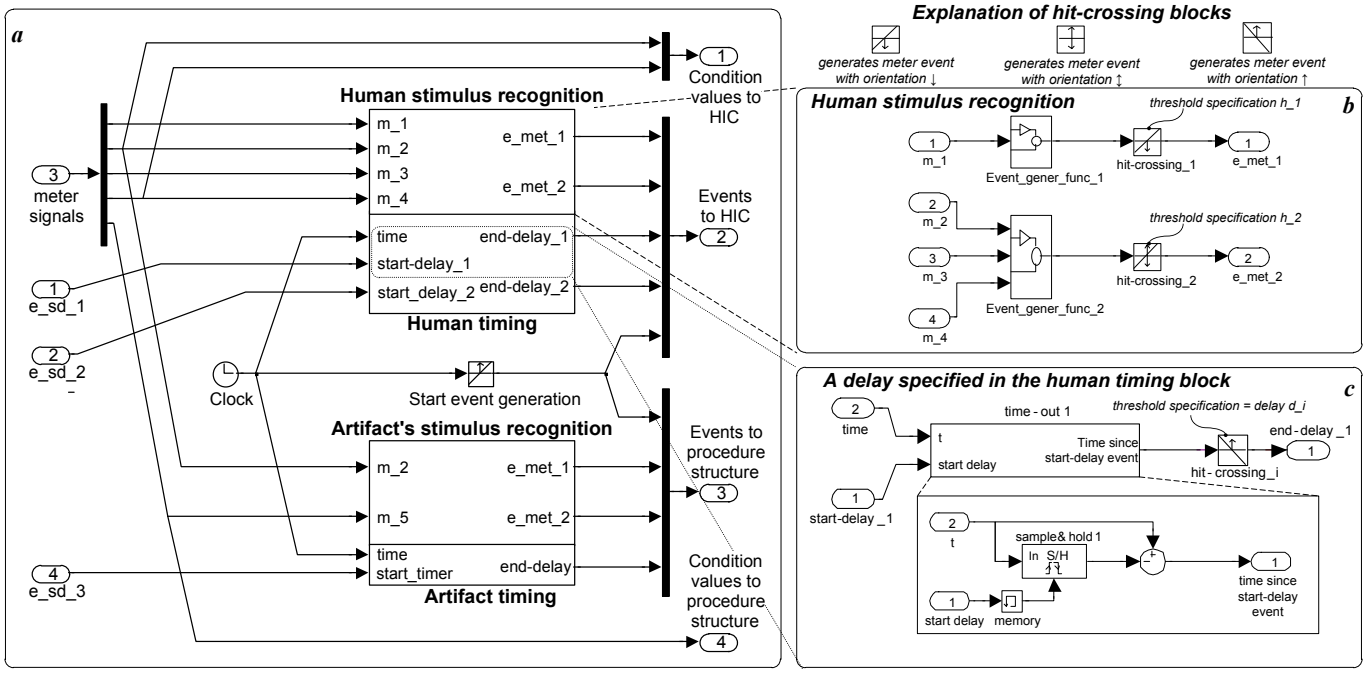


Figure 12. a. Example implementation of a signal conversion specification in Simulink; b. Contents of the human stimulus recognition block; c. Specification of a delay. Numbers of variables, etc., chosen arbitrarily.

of the start event is instantiated in a similar way, using a clock signal as its input. It is shown in the center of Figure 12a. The threshold value specified for the start event corresponds to a delay in seconds between the user-generated start command for the simulation, and the actual start triggered by the start command[9].

The *specifications for timing* define the signal processing for the generation of delays. The specification is instantiated as follows (Figure 12c). When the start-delay event is received, a `time-out` block (which is the equivalent of an event-generating function) starts counting the time since the start-delay event, which is led through a hit-crossing block that generates e_{ed} with $E_{ed} = \{0,\uparrow, d_i, t\text{-}t_{sd}\}$ when the delay has elapsed[10].

5.5 Interfacing control and simulation

Control over the simulation model in Adams is specified as interfacing with Simulink through control values and meter values. Within the simulation model we had to specify how a control value (typically a prescribed angular velocity ω_i) effectuates contractions of a muscle. Control values and meter values to be exchanged with other software are specified using the Adams/Controls module, for instance that a control (input) value $\omega_{prescribed}$ and a meter (output) value $s_{measured}$ (e.g., a distance) are to be exchanged with (Matlab) Simulink is written in Adams modeling language as:

```
variable modify &
 variable_name = .MODEL_1.Controls_Plant_i.input_channels &
 object_value = &
 .model_1.omega_prescribed
!
variable modify &
 variable_name = .MODEL_1.Controls_Plant_i.output_channels &
 object_value = &
 .model_1.s_measured
!
variable modify &
 variable_name = .model_1.Controls_Plant_i.target &
 string_value = "MATLAB"
!
```

Now let us consider how the angular velocity is prescribed to make a limb rotate around its joint by muscle force. Since Adams multibody simulation is based on forward dynamics, we have to translate $\omega_{prescribed}(t)$ to a prescribed force $F_{muscle}(t)$. This is done using a proportional-integral controller:

$$F_{muscle}(t) = K_P\{\omega_{prescribed}(t) - \omega_{actual}(t)\} + K_I \int_0^t \omega_{prescribed}(t) - \omega_{actual}(t)dt$$

where $\omega_{actual}(t)$ is the actual angular velocity (measured feedback). The proportional gain K_P and integral gain K_I are constants that determine the responsiveness and stability of the controller behavior, for which we established adequate values by trial-and-error. In Adams modeling language the above force computation is written as:

```
part create equation differential_equation &
     differential_equation_name = omega_error_integral_i &
     Adams_id = 1 &
     initial_condition = 0.0 &
     function = "omega_prescribed - omega_actual" &
     implicit = off &
     static_hold = off

force create direct single_component_force &
```

[9] This delay can be arbitrarily small, but it cannot be completely avoided. Time is a monotonically increasing function, so that the orientation of the start event must be $r = \uparrow$. If the delay would be set to zero, no threshold is crossed because there is no simulation history before the start command.

[10] The time-out block uses a sample&hold block, which outputs the time signal, but 'freezes' its value when the start delay event is received. The actual simulation time minus this output value equals time elapsed since the start-delay event. We have inserted memory blocks to break algebraic loops. A Memory block outputs its input from the previous time step

```
single component force name = F muscle &
Adams id = 2 &
type of freedom = translational &
i marker name = MARKER i &
j_marker name = MARKER j &
action only = off &
function = "proportional gain * (omega prescribed i-omega i) +
integral gain * DIF(omega error integral i)"
```

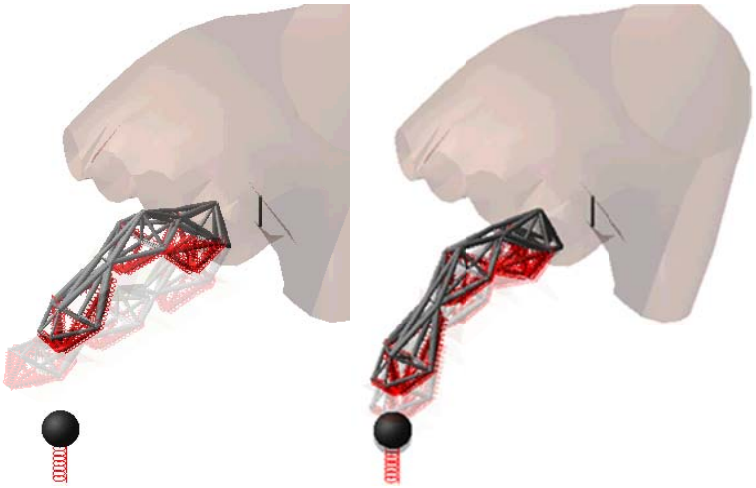
where `MARKER i` and `MARKER j` denote the attachment points of the muscle. In our simplified model of low-level human motion control we allowed muscles not only to contract but also to expand, so that antagonists did not have to be modeled and separately controlled.

In the proof-of-concept implementation, the proxies that we have elaborated in 5.4 are brought together with the simulation model and connected in a Simulink block diagram. We will give an example in the next section. Once the block diagrams and its (sub-)components have been completed, a simulation can be started by clicking on the ► button in any of the Simulink windows. The simulation runs until the total simulation time has elapsed, which is specified by the user in the Simulink block diagram window. There are various options for the visualization of simulations and simulation results. These are briefly discussed and demonstrated in the next section.

6 APPLICATION OF THE PROOF-OF-CONCEPT IMPLEMENTATION

6.1 Objectives and approach

Based on the proxy implementation of the theoretical conceptual elements discussed in the previous sections we elaborated a series of sample cases for experimental application and testing of the proposed approach according to the objectives stated in 5.1. To that end, we applied a strategy of component-based application development. First, we developed four generic sample cases in which we applied the approach to elementary interactions: (i) dropping an object, (ii) throwing an object, (iii) reaching for an object at a given location, (iv) pushing and releasing a foot pedal, (v) pushing and releasing a button, and (vi) grasping, lifting, and carrying an object. These components were defined to be reusable in other contexts and application cases. The six component cases resulted in basic scenarios with only a few transitions, which could be used to simulate the elementary interactions. To be able to test the applicability of the approach in the case of more sophisticated scenarios, we developed three composite cases in which elementary interaction components have been combined in a use process of a product.



a *b*

Figure 14. Scenario-based simulation of a basic interaction: *a.* Positioning the fingertip above a button *b.* Pushing and releasing the button.

In the first composite case, we have combined interactions (i) and (iv) to simulate the use of a pedal bin. In the second composite case, we have combined interactions (ii) and (iii) to simulate a human trying to throw an object into an open garbage bin, and reaching for the object if it lands outside the bin. By varying prescribed velocities in the scenario bundle, we could simulate different paths through the bundle. These first two composite cases [50, 51] were based on early, slightly different versions of the theoretical and implementation elements described in

sections 3-5.

In the third composite case, we brought together interactions (iii), (v), and (vi) to simulate a human customer retrieving a can from a can dispenser after having pushed a button. This composite case, and the component cases brought together in it, have been fully modeled and specified with the theoretical and implementation elements described in sections 3-5. Since the complete can dispenser example is already covered by previous publications

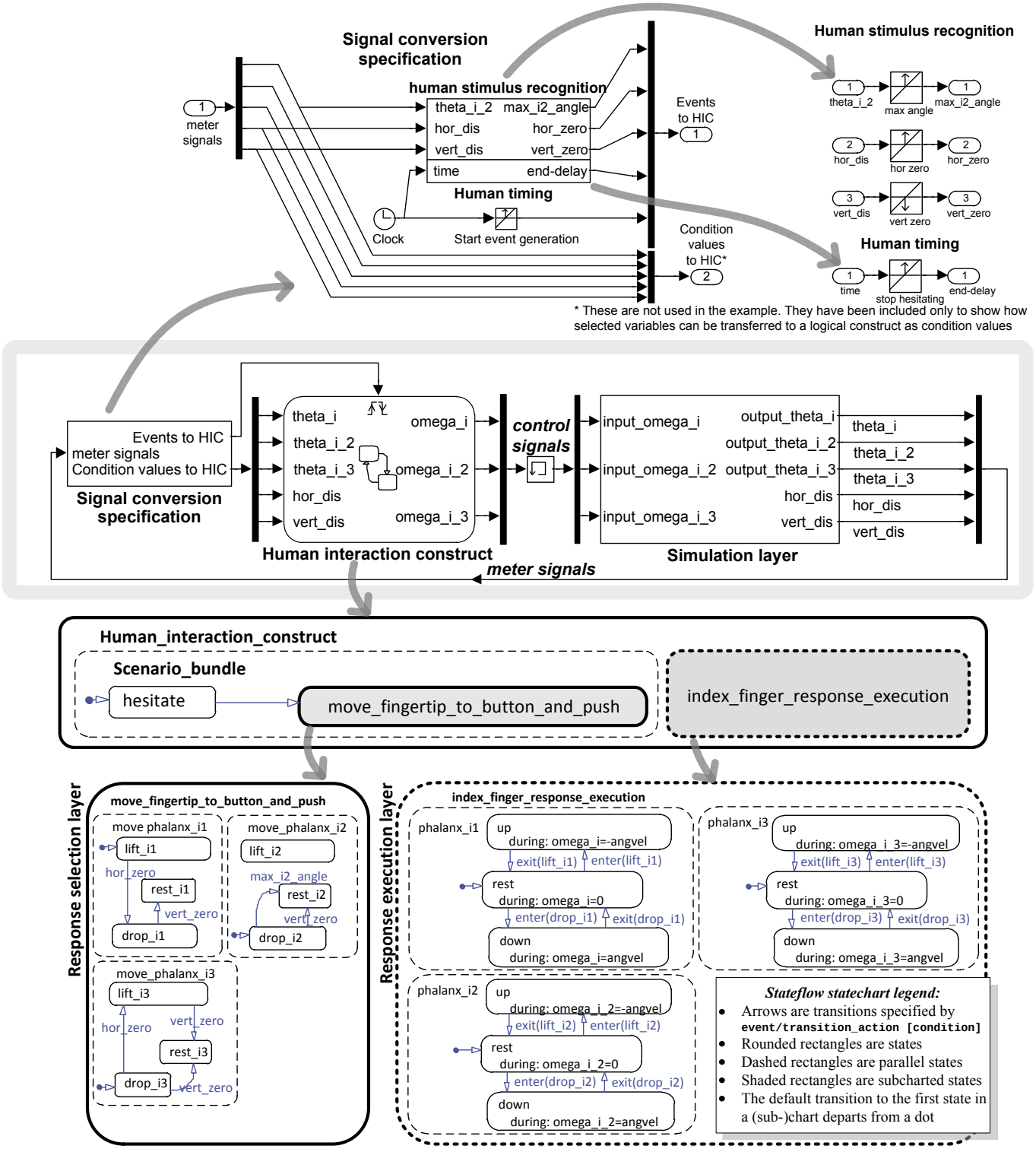


Figure 15 Implementation of button pushing in Simulink: decomposition of models and constructs.

[40, 52], we will now discuss an application case that is simpler and therefore lends itself better for explaining the key theoretical elements that form the focus of this article. It concerns a button with a spring, which is vertically pushed by an index finger (Figure 14). Since, on the other hand, a more comprehensive use process better demonstrates the utility of scenario bundle-based simulation in the practice of design, we will highlight some key outcomes of the can-dispenser case at the end of this section. In that case, horizontal pushing of a similar button appears as a partial interaction.

6.2 Preparing models and specifications for simulation of human-artifact interaction

We created the simulation model in Figure 14 with MSC Adams. It contains a hand fixed to the coordinate system, with an index finger consisting of three phalanges connected by rotational joints. To enable simulation of flexible tissues, we have built up the phalanges from particles connected by spring-dampers (the 'trusses' in Figure 14) and connected them by linear actuators acting as muscles. These react to prescribed angular velocities (relative between consecutive phalanges) that we have specified as control variables.

Figure 15 shows all the models and constructs in Simulink. The complete interfaced control and simulation model is shown in the gray box. The 'simulation layer' block provides the real-time computational connection with the Adams MBS simulation. We have specified the interaction as follows.

After the start event, there is a hesitation (waiting state) that lasts for a given time after $t = 0$, and therefore needs no start-delay event. The end-delay event of the hesitation sets the actual interaction in motion: the finger is brought from its original stretched position to a bent position where its tip is straight above the button (Figure 14a), and then the finger is bent down to touch the button and further press it over a certain distance (Figure 14b). During this last part of the movement, the interaction is actually physical. The two key events based on which this simple interaction is controlled are (i) e_1= `hor_zero`, expressing that the horizontal distance `hor_dis` between the fingertip and the center of the button crosses the value `hor_dis` = 0, and (ii) e_2= `vert_zero`, expressing that the button has been pushed far enough. Until e_1, the first phalanx is lifted while the others are dropped, while between e_1 and e_2, the first and second phalanges are dropped and the third one is lifted. After that, all motion stops (end of the scenario). We took the screenshots in Figure 14 from a successful simulation run controlled by an execution of this scenario. As Figure 15 shows, the path through the use process at scenario-bundle level is strictly linear, and the example does not demonstrate scenarios with multiple options (branching with multiple transitions to and from states, as well as loops) like the composite case that is briefly discussed in the next section does. However, since our definitions of levels of human control have no meaning in terms of statechart representation (where in fact the scenario bundle and the response execution appear at the same level of decomposition), we have chosen to demonstrate that at least the principle works by including some simple branching and loops at the 'lower' levels of response selection and execution specification.

7 DISCUSSION

The example elaborated in the previous section demonstrates the key principles of the theory and proof-of-concept implementation. We have shown how the requested functionality can be concretely realized using commercial off-the-shelf software. We largely had to create constructs and models manually. This involved several activities that can be automated if a dedicated system is developed based on the theory: we had to (i) specify meter values and control values in Ψ, then again in Ξ and in λ_h, (ii) specify events in Ξ (with thresholds and links to meter values) and then again in λ_h to link them to transitions, and (iii) manually 'draw' all the connections for signal exchange between the various constructs. Several of these steps are repetitive and redundant, and in a final dedicated implementation of the theory into a design-supporting system they can obviously be automated. Also, the ultimate

system might use different representations. For instance, it might not be necessary to actually show the designer the system components of Figure 9 and how they are connected, and other FA representations might prove easier to work with than statecharts are.

Even if working with the ultimate system, designers who want to take advantage of the novel functionality of investigating connected interactions, still need to spend some extra time and effort if compared to preparing conventional simulations with a CAD-based model. However, as was suggested above, they do not have to do all the work that we had to do in preparing the models and constructs discussed in this article. Figure 16 shows all the activities and processing steps involved in scenario bundle-based simulation, with omission of the automation opportunities already mentioned above. In operationalizing the proof of concept we had to perform all the activities (or workarounds thereof) except the ones exclusively allocated to the system . The shaded area contains the activities to be performed by the user of the ultimate system. They include activities that are also needed to realize conventional simulations, such as preparing CAD models and simulation models. Furthermore they include optional activities for creating a procedure structure. These are activities that can actually be considered already part of the conventional workflow for products with embedded software [cf. 53]. What remains as extra activities is (bold print in the figure) the specification of a scenario bundle (first as an informal draft, then as a formal control mechanism), selecting targets of successive human movements (e.g., move hand to button, push button till end position, etc.), and specification of event parameters.

In [5] we theoretically evaluated the preparation effort needed for scenario bundle-based simulation by benchmarking it against the preparation effort needed for modeling and simulating connected interactions the traditional way using conventional engineering simulation software, which is by connecting the individual interactions using intermittent constraints [54]. This approach considers the points in time at which parameters in

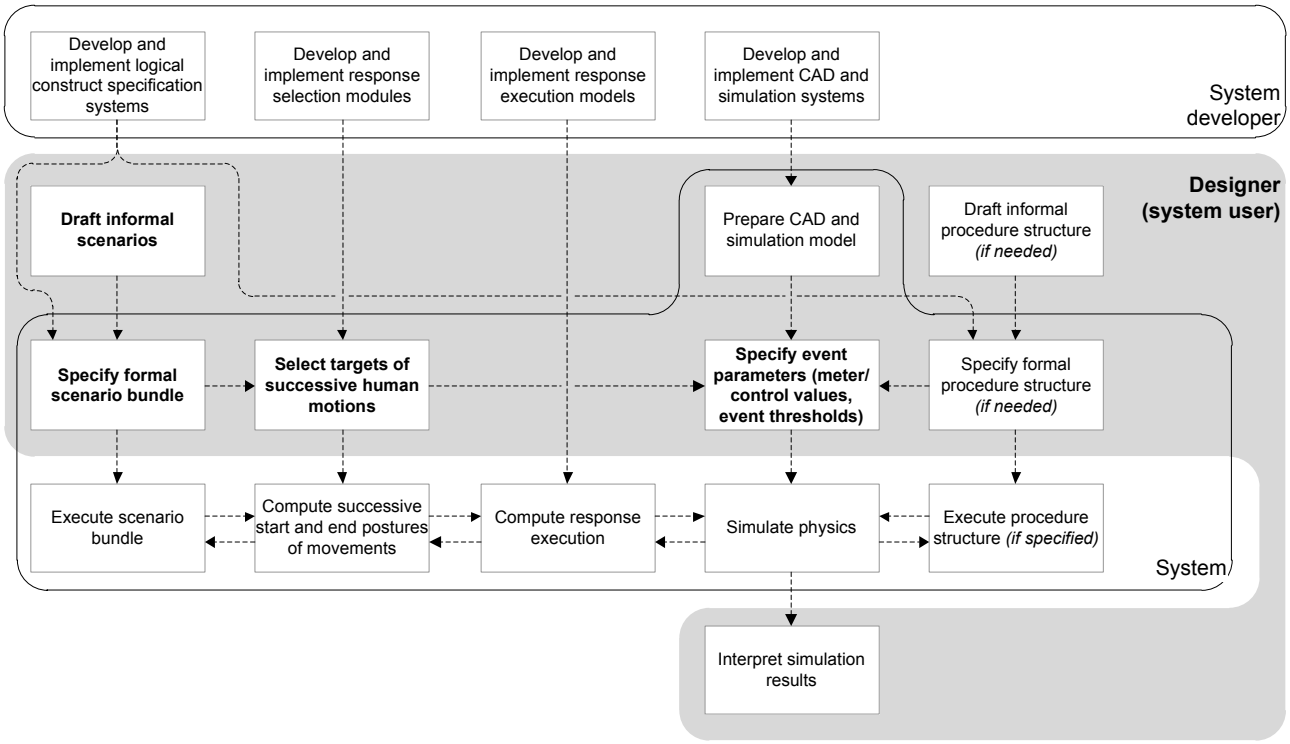


Figure 16. Overview of activities and processing steps involved in scenario bundle-based simulation, with, for the ultimate system realization, activity allocation to system developer, designer (shaded area), and system. The activities printed in bold correspond to the extra workload that is needed for scenario bundle-based simulation.

functions describing the course of the control variables change their value (i.e., changes that correspond to transitions in logical constructs). In terms of computation, it means that changes in meter variables are not evaluated, and no logic is applied. The fact that transition times are not known beforehand implies that all transition times have to be found systematically by running repeated simulations: in order to find the time $t(n)$ of the nth transition, a simulation must be run in which all the preceding transitions $t(0)$,.., $t(n\text{-}1)$ have already been specified. To define consecutive transitions for the same control parameter at multiple points in time, the relative changes must be superimposed by linear addition.

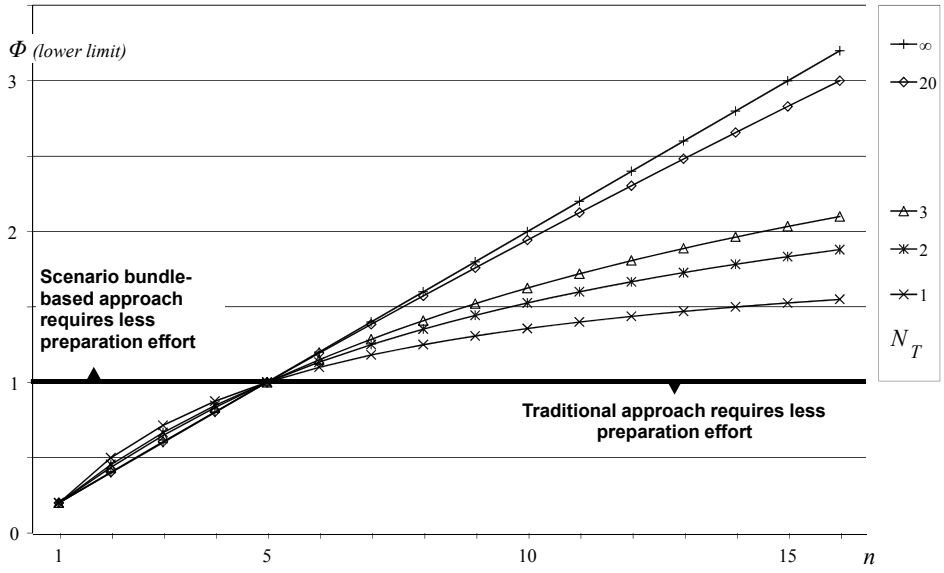


Figure 17. Preparation efforts indicator Φ as a function of the number of model variations, n, and the number of investigated transitions N_T.

The result is a sequential control instruction for *one scenario*, which is *specific for one variation* of the artifact/user model. Any change in these models may result in different transition times, meaning that the whole procedure must be repeated. This already indicates that the extra preparation effort needed for the scenario bundle-based approach might pay off as designers investigate (i) scenarios with more transitions and/or (ii) multiple variants of the artifact/user model with the same scenario. By reasoning based on simulating one series of N_T subsequent transitions and making justifiable assumptions relating the numbers of states, events, and meter signals to the number of transitions, we arrived at the following expression as an approximation for the so-called preparation efforts indicator $\Phi = t_s / t_t$, expressing the ratio between the preparation time t_t needed for the traditional benchmark simulation and t_s for scenario bundle-based simulation:

$$\Phi \geq \frac{nN_T + n - 1}{5N_T + n - 1}$$

with n the number of variations of the artifact/user model being investigated, and $\Phi > 1$ meaning that the scenario-bundle based approach costs less preparation effort. The graph in Figure 17 shows the approximate minimum value of Φ as a function of n for different values of N_T. Since Φ is expressed as an inequality, the possible values of PEI are on *or above* the lines in Figure 17. The scenario bundle-based approach requires less preparation effort in those cases where about five or more model variations are investigated. The scenario-bundle based approach is also advantageous with respect to the number of transitions in the investigated scenario, but only up to a certain limit. For large values of N_T, the minimum value of Φ approaches $\Phi \approx n / 5$.

Although the simplicity of the example in Section 6 is convenient for demonstration purposes, it does not do justice to the potential of the theory when it is used with more complex scenario bundles, more complex interactions, and more complex products. Most importantly, the example did not address the possibility to (i) bring together multiple simulatable scenarios into one bundle, and (ii) include embedded artifact control specified as a procedure structure.

To give an impression of these possibilities, Figure 18 shows the scenario bundle and the simulation model of the

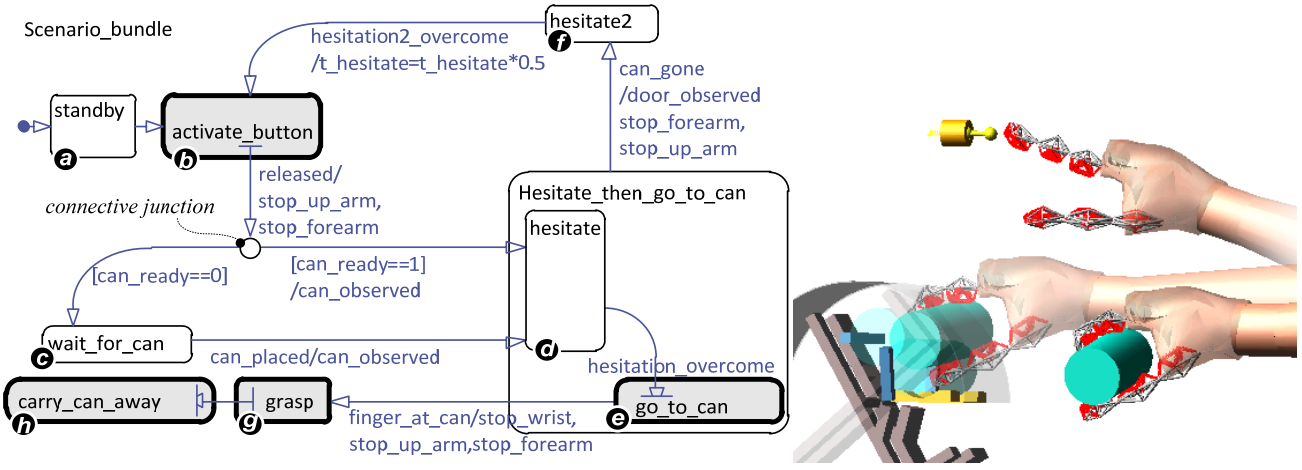


Figure 18. Can dispenser. *a.* Scenario bundle. *b.* compilation of simulation frames

aforementioned can dispenser as presented in [40, 52]. The scenario bundle holds multiple interaction sequences, such as *abdegh, abcdegh, abcfbcdegh,*and *abcdefbcdegh,* all of which we could simulate. The dispenser was controlled by a procedure structure (not shown here) to manage the opening of a protective door and the release of the can. Another extension was the possibility to vary the speed of human movements (fast, medium, slow, etc.), and the human model was extended to a complete arm.

The elaboration of the can dispenser example also revealed stability issues related to our implementation of particle systems in physics simulation, which we introduced to enable investigation of large deformations. These issues significantly contributed to the long time we had to spend fine-tuning state transitions in the response execution layer. Although we found that the theory of scenario bundle-based control can be successfully operationalized in simulations of human-artifact interaction, we had to conclude that before we can develop more example cases to test the integrated approach of controlled simulations, the issues encountered in physics simulation have to be resolved.

To show the utility of the implemented system for product designs and to assess the convenience of use from technical aspects, the proof-of-concept implementation was validated in [5]. It was shown that by scheduling multiple interactions in scenario bundles, time savings can be achieved in preparation of simulations and in computation time. The reason is that with conventional simulation approaches, designers have to prepare and run separate simulations for each consecutive individual interaction, and that preparation work must be repeated for a simulation of the same interaction with a variation of the product design. This is not necessary for scenario bundle-based simulation. Even without reciting the mathematical underpinning in [5], it can be seen that the advantage over conventional approaches increases with the number of consecutive interactions (or more precisely, transitions between interactions) and with the number of design variations that is investigated.

8 CONCLUSIONS AND FURTHER WORK

In this article we have purposefully derived the theoretical elements that are needed to achieve control over physics simulation of human-artifact interaction. The theory explains the principles underpinning the logistics layer containing the scenario bundle, which is a coherent set of scenarios acting as a control mechanism over physics simulation of human-artifact interaction. This makes it possible to perform simulations of procedurally disjunct sequences of interactions based on the designer's conjecture of human decision-making. Based on the theory, we have elaborated a proof-of-concept implementation in which software components for storing and processing constructs, models, and specifications have been realized and successfully applied to simulations of basic

application cases. This elaboration also addresses the arrangement of, and data exchange between the software components in a way that unifies the processing scheme in Figure 3 and the reasoning model of human-artifact interaction in Figure 2.

Scenario bundle-based simulations allow concatenation of multiple interactions in one run without a human subject in the simulation loop and (thus) without having to deploy VR or haptics-based equipment. In comparison to conventional approaches, this presents time savings to designers, especially if the number of sequential interactions is large and if the same set of scenarios is investigated for multiple variations of a product design. Since they only address the higher levels of human decision-making, scenarios can in principle be used without bothering the designer with specification of low-level human motion control. Considering the various opportunities to reduce the specification and modeling time, our goal is that in the ultimate implementation of scenario-bundle based logical control the designer does not need to bother with response execution and only has to completely specify the scenario bundle, thereby choosing predefined modules for response selection, and the procedure structure. We expect that the effort of specifying a scenario bundle will be comparable to the effort of drafting statecharts for specifying the embedded software of a product (i.e., in our terminology the procedure structure), which is currently a routine system-engineering task.

We have successfully operationalized the theory in a proof-of-concept implementation. In testing this application, the real bottle-neck proved to be in the stability of the physics simulation which was compromised by our implementation of simulating large deformations. This issue must be resolved in the further development of a scenario-based simulation system. What is also missing is the incorporation of algorithms that ensure realistic simulation of low-level human motion control and durations of cognitive processes. To that end, we have to investigate the possibility to adopt the algorithms that have recently been developed by others, namely, constraint-based posture prediction, optimization-based motion trajectory computation, and simulation of cognitive architectures. Finally, for the realization of a full-fledged simulation system, a dedicated interface must be developed to (i) specify logical constructs, (ii) link the control-related constructs to each other and to simulations to eliminate the repetitive input that was needed for the proof of concept, and (iii) run simulations and present simulation results.

Apart from these activities, building the ultimate design support system also requires further verification and validation efforts. This includes evaluation of the utility of the system by trials with designers and comprehensive testing with a large number of case studies.

REFERENCES

[1] R.W. Pew, A.S. Mavor, Human-system integration in the system development process: a new look, The National Academic Press, Washington, DC, 2007.

[2] D.W. Carruth, V.G. Duffy, Towards integrating cognitive models and digital human models, in: 11th International Conference on Human-Computer Interaction, Las Vegas, 2005.

[3] I. Horváth, W.F. Van der Vegte, Nucleus-based product conceptualization - Principles and formalization, in: ICED, Stockholm, Sweden, 2003.

[4] Z. Rusák, I. Horváth, W.F. Van der Vegte, First steps towards an all embracing relations-based modelling, in: ASME-DETC. Salt Lake City, UT, 2004.

[5] W.F. Van der Vegte, Testing virtual use with scenarios (PhD thesis), VSSD, Delft, 2009.

[6] H. Van Welbergen, B. Van Basten, A. Egges, Z. Ruttkay, M. Overmars, Real Time Animation of Virtual Humans: A Trade-off Between Naturalness and Control, Computer Graphics Forum, 29 (2010) 2530-2554.

[7] J. Savin, Digital human manikins for work-task ergonomic assessment, Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 225 (2011) 1401-1409.

[8] J.Z. Yang, S. Rahmatalla, T. Marler, K. Abdel-Malek, C. Harrison, Validation of predicted posture tor the virtual human Santos (TM), Digital Human Modeling, 4561 (2007) 500-510.

[9] R. Feyen, L. Liu, D. Chaffin, G. Jimmerson, B. Joseph, Computer-aided ergonomics: a case study of incorporating ergonomics analyses into workplace design, Applied Ergonomics, 31 (2000) 291-300.

[10] A. Veloso, G. Esteves, S. Silva, G. Ferreira, F. Brandão, Biomechanics modeling of human musculoskeletal system using Adams multibody dynamics package, in: 24th IASTED International Conference on Biomedical Engineering, Innsbruck, 2006, pp. 401-407.

[11] M. Damsgaard, J. Rasmussen, S.T. Christensen, E. Surma, M. de Zee, Analysis of musculoskeletal systems in the AnyBody Modeling System, Simulation Modelling Practice and Theory, 14 (2006) 1100-1111.

[12] N.I. Badler, R. Bindiganavale, J.M. Allbeck, W. Schuler, L. Zhao, M.S. Palmer, Parameterized Action Representation for Virtual Human Agents, in: J. Cassell, J. Sullivan, S. Prevost, E. Churchill (Eds.) Embodied conversational agents, MIT Press, Cambridge, MA, 2000, pp. 256-284.

[13] B. Singh, B. Hicks, A.J. Medland, G. Mullineux, J.M. Molenbroek, A contraint based human model for simulating and predicting postures, in: TMCE, Ancona, 2010, pp. 221-229.

[14] K. Abdel-Malek, J.Z. Yang, J.H. Kim, T. Marler, S. Beck, C. Swan, L. Frey-Law, A. Mathai, C. Murphy, S. Rahmatallah, J. Arora, Development of the virtual-human santos (TM), Digital Human Modeling, 4561 (2007) 490-499.

[15] W. Park, D.B. Chaffin, B.J. Martin, J. Yoon, Memory-based human motion simulation for computer-aided ergonomic design, IEEE Transactions on Systems Man and Cybernetics Part A-Systems and Humans, 38 (2008) 513-527.

[16] S.T. Grafton, A.F. Hamilton, Evidence for a distributed hierarchy of action representation in the brain, Human Movement Science, 26 (2007) 590-616.

[17] B. Robbins, D. Carruth, A. Morais, Bridging the gap between HCI and DHM: the modeling of spatial awareness within a cognitive architecture, Lecture Notes in Computer Science, 5620 (2009) 295-304.

[18] J.R. Anderson, D. Bothell, M.D. Byrne, C. Lebiere, An integrated theory of the mind, Psychological Review, 111 (2004) 1036-1060.

[19] D.D. Salvucci, Modeling driver behavior in a cognitive architecture, Human Factors: The Journal of the Human Factors and Ergonomics Society, 48 (2006) 362.

[20] A. Lüdtke, J. Osterloh, Simulating Perceptive Processes of Pilots to Support System Design, Lecture Notes in Computer Science, 5726 (2009) 471-484.

[21] M.D. Byrne, Cognitive architectures in HCI: Present work and future directions, in: Int. Conf. on Human Computer Interaction, Las Vegas, 2005.

[22] D.W. Carruth, M.D. Thomas, B. Robbins, A. Morais, Integrating perception, cognition and action for digital human modellling, Lecture Notes in Computer Science, (2007) 333-342.

[23] N.A. Stanton, C. Baber, A systems analysis of consumer products, in: N.A. Stanton (Ed.) Human factors in consumer products, Taylor & Francis, London, 1998, pp. 75-90.

[24] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima, C. Chen, Formal approach to scenario analysis, IEEE Software, 11 (1994) 33-41.

[25] M. Glinz, An integrated formal model of scenarios based on statecharts, Lecture Notes in Computer Science, 989 (1995) 254-271.

[26] T.A. Alspaugh, D.J. Richardson, T.A. Standish, Scenarios, state machines and purpose-driven testing, in: 4th International Workshop on Scenarios and State Machines, 2005.

[27] B. Cheng, J. Atlee, Research directions in requirements engineering, in: 29th Int. Conference on Software Engineering, IEEE, Minneapolis, 2007, pp. 285-303.

[28] D. Harel, Statecharts: a visual formalism for complex systems, Science of Computer Programming, 8 (1987) 231-274.

[29] C.A. Petri, Fundamentals of a theory of asynchronous information flow, in: 1962 IFIP Congress, Amsterdam, 1962, pp.

386-390.

[30] D. Amyot, A. Eberlein, An Evaluation of Scenario Notations and Construction Approaches for Telecommunication Systems Development, Telecommunication Systems, 24 (2003) 61-94.

[31] M. Elkoutbi, I. Khriss, R.K. Keller, Generating user interface prototypes from scenarios, in: IEEE International Symposium on Requirements Engineering, Limerick, 1999, pp. 150-158.

[32] T.H. Tse, L. Pong, Examination of requirements specification languages, Computer Journal, 34 (1991) 143-152.

[33] K. Perlin, A. Goldberg, Improv: a system for scripting interactive actors in virtual worlds, in: SIGGRAPH, 1996, pp. 205-215.

[34] J. Cremer, J. Kearney, Y. Papelis, HCSM: a framework for behavior and scenario control in virtual environments, ACM Transactions on Modeling and Computer Simulation, 5 (1995) 242-267.

[35] H. Hou, S. Sun, Y. Pan, Research on virtual human in ergonomic simulation, Computers & Industrial Engineering, 53 (2007) 350-356.

[36] D. Harel, A. Pnueli, J.P. Schmidt, R. Sherman, On the formal semantics of statecharts, in: Annual IEEE Symposium on Logic in Computing, 1987.

[37] W.F. Van der Vegte, A survey of artifact-simulation approaches from the perspective of application to use-processes of consumer durables, in: I. Horváth, Z. Rusák (Eds.) TMCE, Ljubljana, Slovenia, 2006, pp. 617-632.

[38] W.F. Van der Vegte, I. Horváth, Including human behavior in product simulations for the investigation of use processes in conceptual design: A survey, in: ASME-CIE, Philadelphia, Pennsylvania, 2006.

[39] W.F. Van der Vegte, I. Horvath, Achieving Closed-Loop Control Simulation of Human-Artefact Interaction: A Comparative Review, Modelling and Simulation in Engineering, 2011, Article ID 675405 (2011).

[40] W.F. Van der Vegte, I. Horváth, Z. Rusák, Simulating the use of products: applying the nucleus paradigm to resource-integrated virtual interaction models, in: I. Horváth, Z. Rusák (Eds.) TMCE, Izmir, 2008, pp. 591-605.

[41] H. Van der Auweraer, Frontloading design engineering through virtual prototyping and virtual reality: industrial applications, in: TMCE, Izmir, 2008, pp. 39-52.

[42] S. Kim, E. Haug, A recursive formulation for flexible multibody dynamics, Part I: open-loop systems, Computer methods in applied mechanics and engineering, 71 (1988) 293-314.

[43] J. Gerstmayr, J. Schöberl, A 3D finite element method for flexible multibody systems, Multibody System Dynamics, 15 (2006) 305-320.

[44] J. Helsen, G. Heirman, D. Vandepitte, W. Desmet, The influence of flexibility within multibody modeling of multi-megawatt wind turbine gearboxes, in: ISMA International Conference on Noise & Vibration Engineering, 2008, pp. 2045-2071.

[45] M. Gonzalez, F. Gonzalez, A. Luaces, J. Cuadrado, Interoperability and neutral data formats in multibody system simulation, Multibody System Dynamics, 18 (2007) 59-72.

[46] S.L. Wang, Motion simulation with Working Model 2D and MSC.visualNastran 4D, Journal of Computing and Information Science in Engineering, 1 (2001) 193-196 ".

[47] Mathworks, Stateflow and stateflow coder for use with Simulink - modeling, simulation, implementation, The MathWorks, Inc., Natick, 2011.

[48] A. Tiwari, N. Shankar, J. Rushby, Invisible formal methods for embedded control systems, Proceedings of the IEEE, 91 (2003) 29-39.

[49] G.E. Stelmach, Information-processing framework for understanding human motor behavior, in: J.A.S. Kelso (Ed.) Human motor behavior - an introduction, pp 63-91, Lawrence Erlbaum Associates, London, 1982.

[50] W.F. Van der Vegte, I. Horváth, Use-driven product conceptualization based on nucleus modeling and simulation with scenarios, in: 15th European Simulation Symposium, Delft, The Netherlands, 2003.

[51] W.F. Van der Vegte, Z. Rusák, Hybrid simulation of use processes with scenario structures and resource-integrated models, in: ASME-CIE, Las Vegas, Nevada, 2007.

[52] W.F. Van der Vegte, Z. Rusák, Controlling simulations of human-artifact interaction with scenario bundles, in:

EDIProD'08, Jurata, 2008, pp. 197-209.

[53] I.R. Kendall, R.P. Jones, An investigation into the use of hardware-in-the-loop simulation testing for automotive electronic control systems, Control Engineering Practice, 7 (1999) 1343-1356.

[54] T. Jia, F.M.L. Amirouche, Optimum impact force in motion control of multibody systems subjected to intermittent constraints, Computers & Structures, 33 (1989) 1243-1249.