Introducing a Formal High-Level Language for Instructing Automated Manikins

Mårdberg P*†, Carlson J.S†, Bohlin R†, Delfs N†, Gustafsson S†, Keyvani A‡, Hanson L§↑

† Fraunhofer-Chalmers Research Centre for Industrial Mathematics, Gothenburg, Sweden

‡ University West, Department of Engineering Science, Trollhättan, Sweden

§ Department of Product and Production Development, Chalmers University of Technology, Gothenburg, Sweden

↑ Industrial Development, Scania CV, Sweden

Abstract

Digital Human Modeling programs are important tools in virtual manufacturing that allow simulation of manual assembly work, long before any physical product has been built. By investigating the logistics, ergonomics and the interaction among the workers at an early stage, it may be possible to reduce the cost of design changes, increase the quality and to decrease the ramp-up time of a manufacturing process.

However, far from all assembly operations are simulated, even if all the necessary data is available. One reason is the tedious work required to setup and to define all the motions needed by a manikin to perform a simulation. In each simulation, the manikin must be adjusted into the desired posture and the user must ensure that balance is held and that it avoids collision with objects in the environment. Thus, even a small case may be time consuming to simulate. This shows that there is a need of an easier way of instructing the manikins.

In this work we propose a new formal high-level language for controlling an automated manikin. The language instructions are structured by a grammar, which defines a hierarchical tree for the manikin instructions. The low-level instructions contains basic functionality for maneuvering the manikin, such as *Move*, *Position* and *Grasp*, and higher levels contain more abstract instructions such as *Get* and *Assemble*. Thus, the high-level instructions define sequences of other instructions, whereas a low-level instruction corresponds to a direct instruction of the manikin.

The set of available instructions that the manikin may perform during a simulation depends on the current state of the manikin and the objects at the assembly station. Thus, properties of objects, such as grasping and mating points, also help define the set of available instructions for the manikin. The order in which the different parts in the assembly operations have to be connected may also be considered when constructing instruction sequences for the manikin. Furthermore, the instruction sequences may be formally verified to ensure that the manikin only performs valid instructions.

The language have been implemented in the manikin simulation software IMMA and tested on elementary cases with relevance to industrial applications. The results show that fewer steps are needed to perform a realistic simulation when the language is used compared to manually instructing the manikin. Furthermore, it is also shown that the instructions generated by the language are formally correct.

Keywords: Digital human modeling, Programming languages, Formal methods

1. Introduction

A Digital Human Modeling program is an important tool in virtual manufacturing which allows simulation of manual assembly work long before any physical product has been built. For instance, it is possible to simulate the material handling, the assembly ergonomics and the interaction among the workers at an early stage of the development. Thus, by resolving design issues and logistic bottlenecks early in the development it is possible to reduce the cost of late design changes, increase the production quality and decrease the

*Peter Mårdberg. Email: peter.mardberg@fcc.chalmers.se

ramp-up time of a manufacturing process (Falck, et al., 2010).
However, far from all assembly operations are simulated even if the all the necessary data is available. One reason for this is the time consuming and tedious work that is required to setup and to define all the motions needed by a manikin to perform a simulation. In each assembly simulation, the user has to position the manikin at the workstation, adjust the manikin into the desired posture and select the correct grip. Moreover, to make the simulation relevant, the user must ensure that the manikin maintains balance during the simulation and that it avoids collision with objects in the environment. Hence, even a small case may be time consuming to simulate since many instructions might be required from the user to create a relevant simulation. Thus, it is well motivated to find an easier way of instructing the manikins.

In this work we propose a new language that uses both high and low-level instructions to control an automated manikin. A grammar structures the different instruction levels into a hierarchical tree where the lowest level in the tree contains basic instructions for maneuvering the manikin, such as *Move*, *Position* and *Grasp*, and the higher levels contain more abstract instructions such as *Get* and *Assemble*. Thus, a high-level instruction such as *Assemble* defines sequences of other instructions, whereas a low-level instruction such as *Grasp* corresponds to a direct instruction to the manikin.

The set of available instructions that the manikin may perform during a simulation depends on the current state of the manikin and on the state of the objects in the assembly station. For instance, if the manikin grasps an object with both hands, it is impossible for the manikin to grasp another object. The properties of objects, such as grasping and mating points also help define the set of available instructions for the manikin. This since each low-level keyword must have a corresponding action in the simulation. A *Grasp* instruction may only be used if there is an object that is available for the manikin to grasp.

It is also possible to consider the order in which the different parts should be assembled when constructing instruction sequences for the manikin. Thus, it is possible to prevent a manikin from performing an assembly instruction unless all the preconditions to that instruction are fulfilled.

All the objects in the assembly station and the manikin are composed into the same discrete model. Moreover, the instruction language is defined to construct the transition sequences in the model. Thus, the language, the assembly station and the manikin use the same discrete computational model. This makes it possible to apply formal methods to verify that the assembly operation is performed correctly, that the model specification is not violated and that no deadlocks occur.

The language has been implemented in the Intelligently Moving Manikins (IMMA) (Hanson, et al., 2011) software application, and it has been tested on elementary cases with relevance to industrial applications. The results show that fewer keywords are needed to make the manikin perform realistic simulations when high-level instructions are used compared to only using low-level instructions. Furthermore, it is also proved that the assembly instructions generated by the language are formally correct.

The main contributions of this paper are, (i) how a formal high-level language may be defined to instruct an automated manikin, (ii) how the instruction language is linked with the objects in the assembly station by showing how the language, the manikin and the objects may use the same discrete computational model, and (iii) that we show with our modeling approach that it is possible to apply formal methods to prove that the assembly operations are correctly performed.

This paper is organized as follows. Section 2 covers the requirements for using the language, whereas Section 3 covers our modeling approach. Section 4 shows some case studies followed by discussion and future work in Section 5. Concluding remarks are found in Section 6.

2. Requirements

In this work we show how to model manikin as a Discrete Event System, and how a high-level language may be used to efficiently instruct the manikin to perform assembly operations. However, our proposed approach requires an automated manikin. We say that a manikin is automated if it is able to automatically perform an assembly operation. Thus, if the manikin is instructed to grasp an object, then it should be able to automatically reposition itself and grasp the object without any further help from the user. Moreover, it is not sufficient for the manikin to just automatically perform the assembly operation; it also needs to maintain balance during the operation. The balance has to consider the body parts and the objects being carried as well as exterior forces and torques from the environment. Furthermore, it also needs to automatically avoid collision with the objects in the assembly station (Bohlin, et al., 2012; Delfs, et al., 2013).

3. Modeling

The instruction language, the manikin and the objects to be assembled have to be composed into the same computational model to be able to efficiently instruct the manikin and at the same time guarantee that the instructions are correctly performed. This makes it possible to apply formal methods to prove that the constructed assembly sequences do not violate any modeled specifications such as assembly order and physical constrains.

This section is divided into three subsections. The first shows how we define the grammar for the high and low-level instructions which control the manikin. This is followed by a section on how we model the objects and the manikin. Finally, we show how our composed model may be formally verified.

3.1. Instruction Grammar

Our proposed language is a context free language where the keywords in the language are divided into high and low-level instructions. The high-level instructions define sequences of other instructions, whereas low-level instructions correspond to direct instructions to the manikin. Since the language is context free, there exists a non-deterministic pushdown automaton (Hopcroft, et al., 2007) that defines a state machine controlling the transitions in the underlying Discrete Event System (DES) (see Section 3.2). Each instruction corresponds to a non-empty set of transitions in the composed model. In this way it is possible to integrate the language into the DES model. Also, due to the formal definition of the language it is possible to prove that specific properties hold for the language.

A grammar is defined to formally structure the instruction levels into a hierarchical tree (Hopcroft, et al., 2007; Aho, et al., 2007). The grammar furthermore defines how the instruction sentences are generated, and hence ensures that all the corresponding arguments are set (Aho, et al., 2007). For instance, consider the instruction $Grip \rightarrow RightHand \rightarrow Object_i \rightarrow TargetPoint_j$. Here $Grip$ is the keyword and $RightHand$, $Object_i$ and $TargetPoint_j$ are the corresponding arguments.

3.2. Modeling Manikins and the Scene Using EFAs

A sequence of assembly instructions is said to be valid if it does not contain any contradictory instructions that violate the model specification. For instance, to grasp an object that currently may not be grasped or to give identical instructions in consecutive order, such as $Release \rightarrow RightHand$ directly followed by $Release \rightarrow RightHand$, are contradictory instructions.

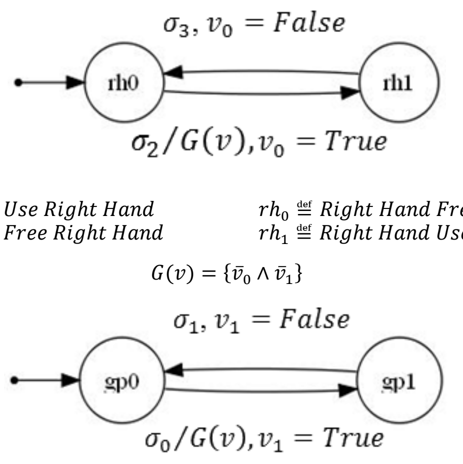
To be able to prove that the generated assembly instructions are formally correct, the manikin has to be modeled into the same discrete event system as all the objects in the assembly operation. The model must also include all the properties of the objects that may directly or indirectly be used in the assembly.

All objects are modeled by an Extended Finite Automaton (EFA). Using the notation in (Bengtsson, et al., 2012), an EFA E might be defined as $E = \langle Q \times V, \Sigma, G, A, \rightarrow, (q_0, v_0) \rangle$, where Q denotes the set of state locations, V is finite set of variables and Σ the non-empty finite set of events. The guard predicates are denoted by G whereas A denotes the action functions that update $v \in V$. The transition relation and the starting state of the automation are denoted $\rightarrow \subseteq Q \times \Sigma \times G \times A \times Q$ and (q_0, v_0), respectively. A transition in E may only occur if the corresponding guard predicate is fulfilled. If the transition occurs, then the corresponding action updates the variable set V (Bengtsson, et al., 2012; Miremadi, et al., 2011).

All objects are modeled separately as EFAs and are composed into the same model using an another EFA, denoted E_{All}, defined as the parallel synchronization of all EFAs in the scene: $E_{All} = E_0 \parallel E_1 \parallel \cdots$ (Sköldstam & Åkesson, 2007; Miremadi, et al., 2008). The composed automaton is automatically constructed from the objects in the scene. Thus, when a manikin is included in the scene, the corresponding EFAs are added to E_{All}. Moreover, when a user defines a grip point on an object, an EFA for that grip point is also added to E_{All}.

Each EFA may contain $a \in A$ and predicates that define illegal states, such as a grip point being used by both hands at the same time. These predicates are used to create guards that prevent these states from occurring. The guards in the EFAs form the model specification for the composed automata E_{All}, and are used to automatically define a set of guard predicates for E_{All} to prevent transitions which violate the model specification. Once all the guards have been added to E_{All}, the set of allowed transitions corresponds to the set of instructions that the user may perform when constructing assembly sequences. Furthermore, the user may add constraints, such as assembly order, into E_{All}, which then also are included in the set of allowed transitions.

The following example shows how a grip instruction may be modeled by a pair of two-state automata, see Figure 1. Let $Q_{rh} = \{rh_0 \stackrel{\text{def}}{=} Right\ Hand\ Free, rh_1 \stackrel{\text{def}}{=} Right\ Hand\ Used\}$ and $Q_{gp} = \{gp_0 \stackrel{\text{def}}{=} Grip\ Point\ Free,\ gp_1 \stackrel{\text{def}}{=} Grip\ Point\ Used\}$ define the states in each automation, and let $\Sigma_{rh} = \{\sigma_2 \overset{\text{def}}{=} Use\ Right\ Hand, \sigma_3 \overset{\text{def}}{=} Free\ Right\ Hand\}$, and $\Sigma_{gp} = \{\sigma_0 \overset{\text{def}}{=} Use\ Grip, \sigma_1 \overset{\text{def}}{=} Free\ Grip\}$ define the events. Moreover, let v be defined as a Boolean string of length 2 as $v \in \mathbb{B}^2$, and let a guard be defined as Boolean function as $G(v) \rightarrow \mathbb{B}$. The Boolean string v and the guard $G(v)$ are shared by both automata. A high-level instruction for this example could be $Grip \rightarrow\ RightHand\ \rightarrow Object_i \rightarrow TargetPoint_j$, and when executed the corresponding events are called in the underlying automata $\{\sigma_0, \sigma_2\}$. If the corresponding guard, $G(v) = \{\bar{v}_0 \wedge \bar{v}_1\}$, evaluates to true then v_0 and v_1 are updated to true, and state transitions are made. Thus, the shared guards G and variables V may be used to control the transitions in the composed automaton.



$\sigma_2 \overset{\text{def}}{=} Use\ Right\ Hand$ $\quad rh_0 \overset{\text{def}}{=} Right\ Hand\ Free$
$\sigma_3 \overset{\text{def}}{=} Free\ Right\ Hand$ $\quad rh_1 \overset{\text{def}}{=} Right\ Hand\ Used$

$G(v) = \{\bar{v}_0 \wedge \bar{v}_1\}$

$\sigma_0 \overset{\text{def}}{=} Use\ Grip$ $\quad gp_0 \overset{\text{def}}{=} Grip\ Point\ Free$
$\sigma_1 \overset{\text{def}}{=} Free\ Grip$ $\quad gp_1 \overset{\text{def}}{=} Grip\ Point\ Used$

Figure 1 Shows a pair of two-state automata. The upper models the grip of the right hand whereas the lower models the state of the grip.

3.3. Formal Methods

Formal verification is a methodology to prove if a set of properties hold for a model (Voronov & Åkesson, 2009), and is used to verify that no language instruction violates our model E_{All} and that there are no contradictions in the model specification.

The compound EFA may be translated into propositional logic and thus also expressed as a Binary Decision Diagram (BDD) (Miremadi, et al., 2012), A BDD is a data structure that allows efficient computation of logic operations (Huth & Ryan, 2004; Andersen, 1997). It forms a tree where each node has two outgoing edges corresponding to if the variable is true or false. The leaf nodes in a BDD are defined to be true and false. Thus, all paths that end in the true-leaf are assignments that satisfy the model whereas each path ending up in the false-leaf is a contradiction to the model (Huth & Ryan, 2004; Andersen, 1997), see Figure 2. By expressing the properties of the manikin and objects as propositional logic it is possible to use them in the verification synthesis. For instance, it is possible to verify if a set of states are reachable from the initial state, or that there exists a way to return to the initial state. In this way, the manikin is prevented from reaching a deadlock state or to violate any properties of the model.

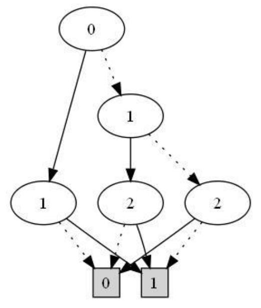


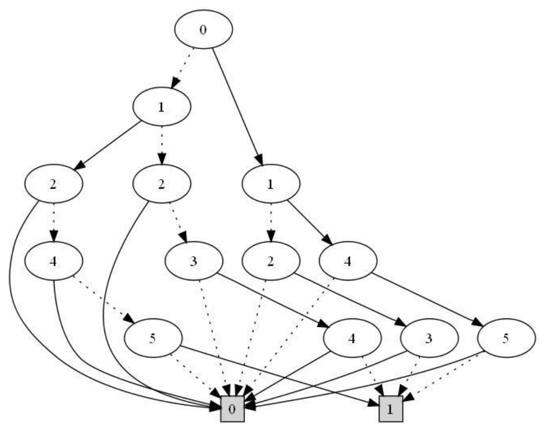
Figure 2 Shows the BDD for $(\bar{\boldsymbol{x}}_0 \wedge \bar{\boldsymbol{x}}_1 \wedge \bar{\boldsymbol{x}}_2) \vee (\boldsymbol{x}_0 \wedge \boldsymbol{x}_1) \vee (\boldsymbol{x}_1 \wedge \boldsymbol{x}_2)$.

4. Case Studies

Here we present three case studies. The cases are small but relevant, and show the advantages of our proposed high-level language. The first two cases show how the model prevents the user from performing erroneous instructions, and how the model may be used to ensure that a specific assembly order is followed. The final test case shows how the language is used to generate an assembly sequence.

4.1. Test Case A

This case illustrates how the EFA model is used to restrict the usage of the hands to one hand at a time. Figure 3 shows the corresponding paths of the states in the BDD, and it may be observed that the events $Use\ Right\ Hand$ and $Use\ Left\ Hand$ mutually exclude each other.



States:

$Right\ Hand\ Used \overset{\text{def}}{=} Node_2$
$Right\ Hand\ Free \overset{\text{def}}{=} Node_3$
$Left\ Hand\ Used \overset{\text{def}}{=} Node_4$
$Left\ Hand\ Free \overset{\text{def}}{=} Node_5$

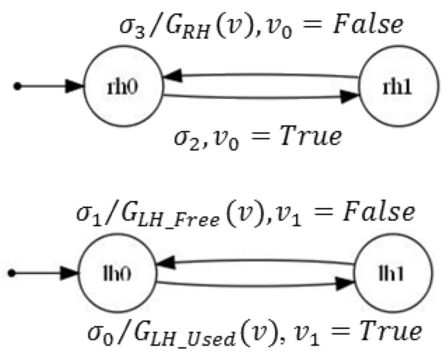
Events:

$Use\ Right\ Hand \overset{\text{def}}{=} \{Node_0 = 0,\ Node_1 = 0\}$
$Free\ Right\ Hand \overset{\text{def}}{=} \{Node_0 = 1,\ Node_1 = 0\}$
$Use\ Left\ Hand \overset{\text{def}}{=} \{Node_0 = 0,\ Node_1 = 1\}$
$Free\ Left\ Hand \overset{\text{def}}{=} \{Node_0 = 1,\ Node_1 = 1\}$

Figure 3 Shows a BDD where it is only possible for a grip to occur with one hand at a time.

4.2. Test Case B

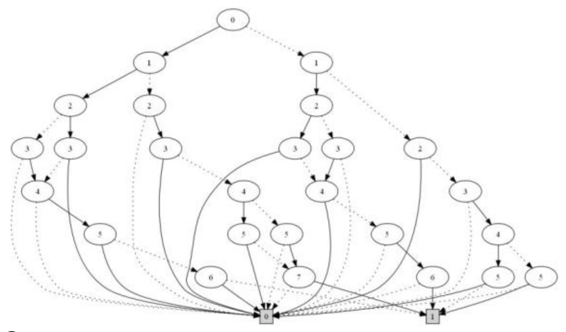
In this case we show how our EFA model might be expanded to include the assembly order of an operation. The assembly order is expressed as a constraint, as for instance in this case, prevents the manikin from grasping an object with the left hand unless the right hand has first grasped the object. Moreover, the left hand cannot release the object unless the right one has released it first. Figure 4 shows the automata for the constraint whereas Figure 5 shows the corresponding BDD of the model.


$\sigma_3/G_{RH}(v), v_0 = False$
rh0
rh1
$\sigma_2, v_0 = True$
$\sigma_1/G_{LH\_Free}(v), v_1 = False$
lh0
lh1
$\sigma_0/G_{LH\_Used}(v), v_1 = True$

States:	***Guards*** :
$rh_0 \overset{\text{def}}{=} Rigth\ Hand\ Free$	$G_{RH}(v) = v_1$
$rh_1 \overset{\text{def}}{=} Right\ Hand\ Used$	$G_{LH\_Used}(v) = v_0$
$lh_0 \overset{\text{def}}{=} Left\ Hand\ Free$	$G_{LH\_Free}(v) = \overline{v_0}$
$lh_1 \overset{\text{def}}{=} Left\ Hand\ Used$	***Variables***:
Events:	$v = \{v_0, v_1\}$

$\sigma_0 \overset{\text{def}}{=} Use\ Left\ Hand$
$\sigma_1 \overset{\text{def}}{=} Free\ Left\ Hand$
$\sigma_2 \overset{\text{def}}{=} Use\ Right\ Hand$
$\sigma_3 \overset{\text{def}}{=} Free\ Right\ Hand$

Figure 4 Shows two automata. The upper models the grip of the right hand whereas the lower models the left hand.



States:

$Right\ Hand\ Used \overset{\text{def}}{=} Node_2$
$Right\ Hand\ Free \overset{\text{def}}{=} Node_3$
$Left\ Hand\ Used \overset{\text{def}}{=} Node_4$
$Left\ Hand\ Free \overset{\text{def}}{=} Node_5$
$v_0 \overset{\text{def}}{=} Node_6$
$v_1 \overset{\text{def}}{=} Node_7$

Events:

$Use\ Right\ Hand \overset{\text{def}}{=} \{Node_0 = 0,\ Node_1 = 0\}$
$Free\ Right\ Hand \overset{\text{def}}{=} \{Node_0 = 0,\ Node_1 = 1\}$
$Use\ Left\ Hand \overset{\text{def}}{=} \{Node_0 = 1,\ Node_1 = 0\}$
$Free\ Left\ Hand \overset{\text{def}}{=} \{Node_0 = 1,\ Node_1 = 1\}$

Figure 5 Shows the BDD for the composed automation in Figure 4.

4.3. Test Case C

This case demonstrates how the language is used to construct an assembly sequence. The manikin picks up a tunnel bracket from a table and brings it into assembly position. The tunnel bracket is then assembled inside the car by following an existing assembly path for the 6DOFs tunnel bracket. Figure 6 shows six frames of the assembly operation and Table 1 shows both the instructions needed for our proposed language and the number of instructions needed by the user to perform the assembly operation without the language. The existing assembly path has been generated in IPS (Industrial Path Solutions, 2012) and is guaranteed to be collision free. Figure 7 shows the dialog window used to generate the assembly instructions used in this case.

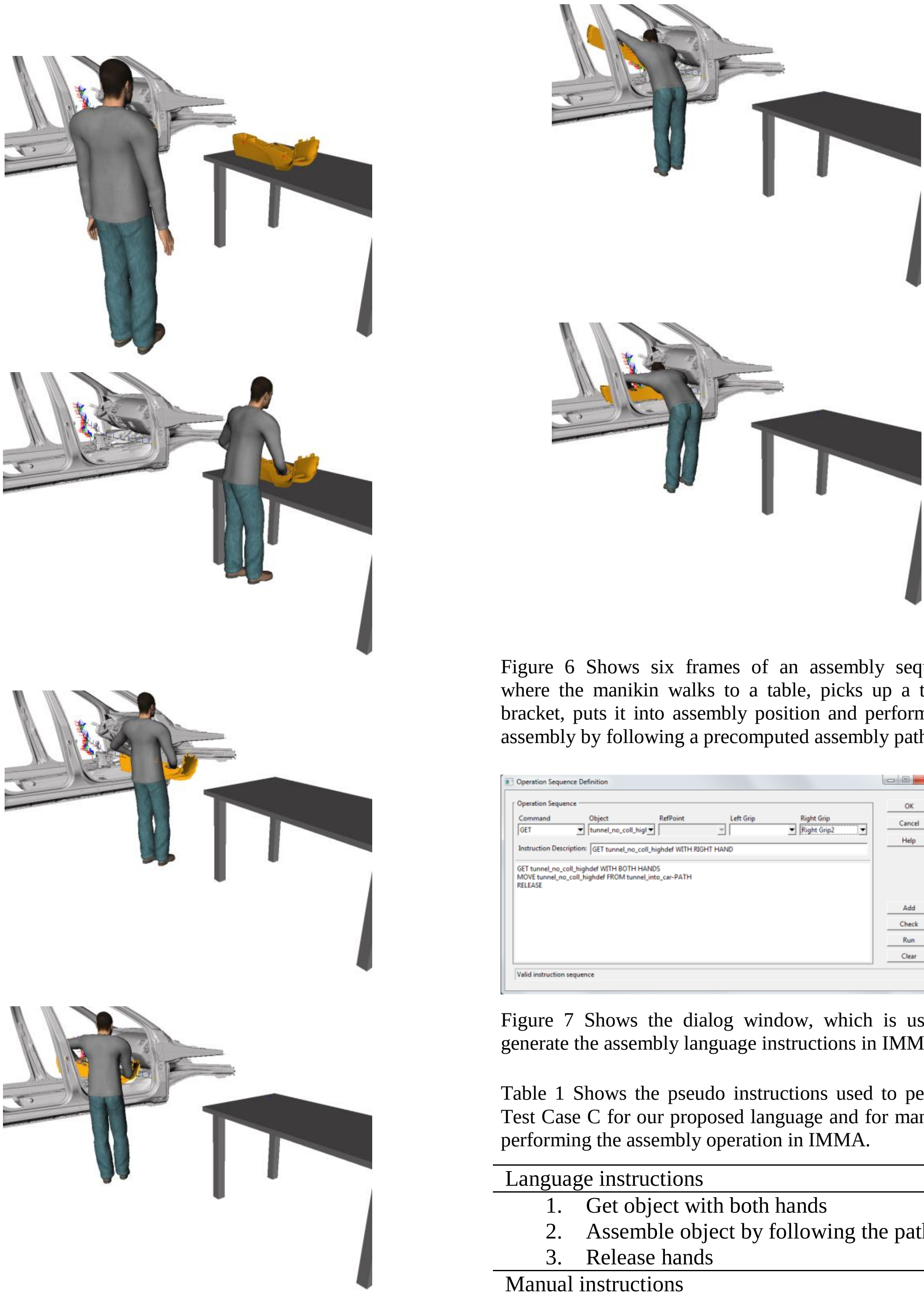
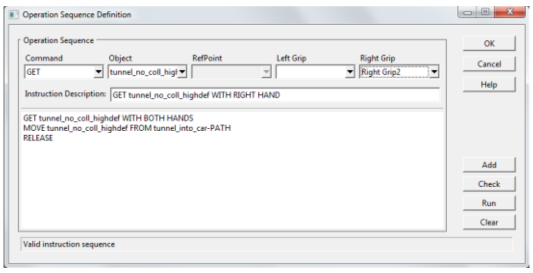


Figure 6 Shows six frames of an assembly sequence where the manikin walks to a table, picks up a tunnel bracket, puts it into assembly position and performs the assembly by following a precomputed assembly path.



Command	Object	RefPoint	Left Grip	Right Grip
GET	tunnel_no_coll_higl			Right Grip2

Instruction Description: GET tunnel_no_coll_highdef WITH RIGHT HAND

```
GET tunnel_no_coll_highdef WITH BOTH HANDS
MOVE tunnel_no_coll_highdef FROM tunnel_into_car-PATH
RELEASE
```

Valid instruction sequence

Figure 7 Shows the dialog window, which is used to generate the assembly language instructions in IMMA.

Table 1 Shows the pseudo instructions used to perform Test Case C for our proposed language and for manually performing the assembly operation in IMMA.

Language instructions	
1.	Get object with both hands
2.	Assemble object by following the path
3.	Release hands
Manual instructions	
1.	Grip right hand
2.	Grip left hand
3.	Move manikin to the assembly path[1]
4.	Connect the object to the path
5.	Follow path
6.	Release hands

[1] This instruction is actually not possible to perform in IMMA without the language, however for comparison, it is assumed to exist.

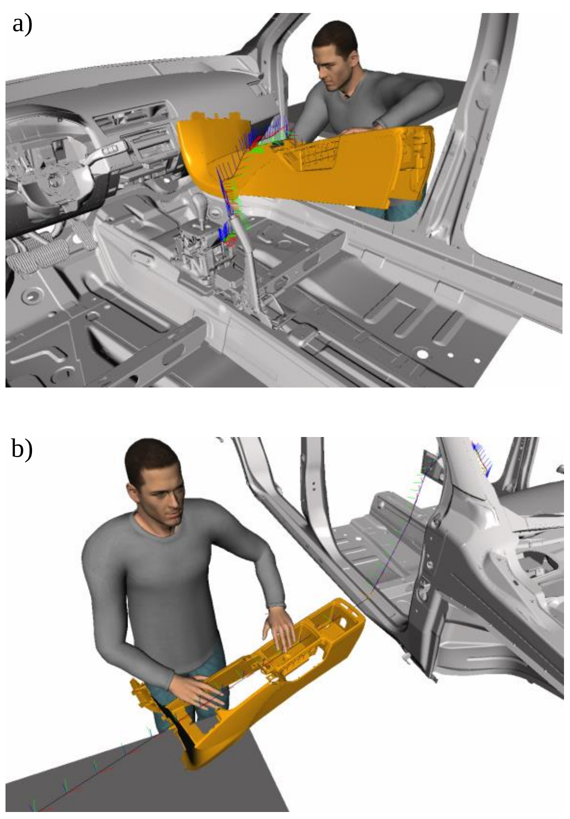


Figure 8 Shows a) the precomputed assembly path, and b) the assembly path that has been automatically generated by the language.

5. Discussion and future work

The provided test cases are small and concise, and they show some of the advantages of our approach to integrating the high-level language with the manikin, the objects and the assembly instructions into the same model.

Fewer instructions are needed when high-level instructions are used. For instance, if the user wants the manikin to grasp an object with the right hand, then the following commands needs to be performed. Firstly, the manikin needs to be reoriented to face the grasping point. Then the user has to define a TCP for the right hand on the manikin to indicate that it should grip, and to select to which grasping point. The next time the manikin is asked to reposition itself, it will grasp the object. These instructions are automatically handled by the language when $Use\ Right\ Hand$ is executed.

The IMMA manikin uses assembly paths to perform the assembly operations. An assembly path for placing a tunnel bracket inside a car is shown in Figure 8 a). Hence, to automate the usage of the manikin, the language constructs new paths between the assembly objects, see Figure 8 b). In fact, it is not possible to perform all assembly sequences without the language, since the language creates some of the assembly paths. However, it is also necessary to use already existing assembly paths since the generated paths are not guaranteed to be collision free.

The proposed language further extends the automated level of the IMMA manikin by taking advantage of properties of the assembled objects and from the assembly instructions. All the objects that are direct or indirectly used in the assembly operation are composed into the same model. Thus, if an object changes state, then the change of state also occurs in all other automata in E_{All}. Hence, a change of state in one automaton may prevent state transitions in others. For instance, by grasping an object with the right hand it may be impossible for the manikin to grasp the same object with the left hand.

Since E_{All} may be expressed as a BDD, it is possible to efficiently verify properties in the model and that the model specification is not violated (Miremadi, et al., 2012). It is for instance possible to prove that a state is reachable from the initial state and that there is a path leading back, which may correspond to prove that a manikin may grab a tool, use it and then return it. In the same manner, it is possible to expand the model with a set of preconditions that prevent states to be reached unless the all of the preconditions are fulfilled, and thereby ensure that an assembly sequence is carried out in a specific order.

A drawback when modeling with automata is the exponential growth of the state space, which can make it infeasible to formally verify the model (Sköldstam & Åkesson, 2007). However, in (Voronov & Åkesson, 2009) it is shown that by utilizing EFA instead of FA, the size of the automata is reduced. Furthermore, (Miremadi, et al., 2012) show that with the same modeling approach it is possible to verify EFA models of industrial size.

In future work, more keywords and instruction levels will be introduced to the language. Also, the possibility to use the instruction sequence together with the positions of the objects, the manikin and a Predetermined Motion Time System (PTMS) will be investigated. In this way, it may be possible to use the data in the PTMS to estimate the time that is needed to perform the assembly sequence. Moreover, it should be possible to use a manikin instruction sequence to generate a worksheet for assembly workers.

6. Conclusion

In this work we present a new high-level language for instructing manikins and test it on assembly cases with relevance to industrial applications. The proposed language is designed to use the automated functions of the IMMA manikin, which reduces the number of instructions needed to generate an assembly simulation. Thus, the user may focus more on what the manikin should perform rather than how it performs it. Furthermore, the language contains different instruction levels, where the lowest level corresponds to specific instructions that allow the user to directly manipulate the manikin.

Moreover, with our modeling approach, the whole assembly operation may be included into the same model as the manikin. This way, it is possible to formally prove that the manikin correctly performs the assembly operation. The modeling approach is general and may be expanded to allow more properties in the objects, more complex assembly instructions and more manikins to be included into the model.

Acknowledgement

This work was carried out within The Swedish Foundation for strategic Research (SSF) ProViking II program and the Wingquist Laboratory VINN Excellence Centre, supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA).

This work is a part of the Sustainable Production Initiative and the Production Area of Advance at Chalmers University of Technology.

References

Aho, A. V., Lam, M. S., Sethi, R. & Ullman, J. D., 2007. *Compilers : principles, techniques, & tools.* Boston: Pearson Addison-Wesley.

Andersen, H., 1997. *An Introduction to Binary Decision Diagrams,* Lyngby, Denmark: Dept. Inf. Technol., Tech. Univ. Denmark.

Bengtsson, K. et al., 2012. Sequence Planning Using Multiple and Coordinated Sequences of Operations. *IEEE Transactions on Automation Science and Engineering,* 9(2), pp. 308 - 319.

Bohlin, R. et al., 2012. *Automatic Creation of Virtual Manikin Motions Maximizing Comfort in Manual Assembly Processes.* Ann Arbor, Michigan, USA.

Delfs, N. et al., 2013. *Introducing Stability of Forces to the Automatic Creation of Digital Human Postures.* Ann Arbor.

Falck, A.-C., Örtengren, R. & Högberg, D., 2010. The Impact of Poor Assembly Ergonomics on Product Quality: A Cost–Benefit Analysis in Car Manufacturing. 20(1), p. 24–41.

Hanson, L., Högberg, D., Bohlin, R. & Carlson, J., 2011. *IMMA – Intelligently Moving Manikins – Project Status 2011.* Lyon, First International Symposium on Digital Human Modeling.

Hopcroft, J. E., Motwani, R. & Ullman, J. D., 2007. *Introduction to automata theory, languages and computation.* Boston : Pearson Addison-Wesley, cop.

Huth, M. & Ryan, M., 2004. *Logic in computer science : modelling and reasoning about systems.* Cambridge : Cambridge Univ. Press.

Industrial Path Solutions, 2012. [Online] Available at: www.industrialpathsolutions.com

Miremadi, S., Åkesson, K. & Lennartson, B., 2008. *Extraction and Representation of a Supervisor Using Guards in Extended Finite Automata.* Göteborg.

Miremadi, S., Åkesson, K. & Lennartson, B., 2011. Symbolic Computation of Reduced Guards in Supervisory Control. *IEEE Transactions on Automation Science and Engineering,* 8(4), pp. 754-765.

Miremadi, S., Åkesson, K. & Lennartson, B., 2012. A BDD-based Approach for Modeling Plant and Supervisor by Extended Finite Automata. *IEEE Transactions on Control Systems Technology,* 20(6), pp. 1421-1435.

Sköldstam, M. & Åkesson, K., 2007. *Modeling of Discrete Event Systems using Finite Automata With Variables.* New Orleans. LA, USA, Proceedings of the 46th IEEE Conference on Decision and Control.

Voronov, A. & Åkesson, K., 2009. *Verification of process operations using model checking.* Bangalore, India, August 22-2, 2009.