Toward Specifying Human-Robot Collaboration with Composite Events

DOI: 10.1109/ROMAN.2016.7745225
Handle: http://hdl.handle.net/1942/21492

Toward Specifying Human-Robot Collaboration with Composite Events*

Jan Van den Bergh[1], Fredy Cuenca Lucero[2], Kris Luyten[1] and Karin Coninx[1]

Abstract— **Human-Robot Collaboration is increasingly considered in manufacturing to better combine the strengths of humans and robots. Establishing this human-robot collaboration may require multi-modal interaction; input to and output from the robot can both use multiple channels in sequence or in parallel. Designing effective interaction requires the expertise from different domains, possibly originating from people with different backgrounds.**

In our work we explore how composite events — hierarchical composition of events — can be used in a way that eases the communication within a multi-disciplinary team. In this paper, we present how the concept of composite events can be used to create different layers of abstraction that can be used to ease prototyping and discussion of human-robot collaboration with stakeholders through a supporting tool called Hasselt UIMS. At the lower level(s) of abstraction, the composite events can be mapped to the message-based communication as implemented in the Robotic Operating System (ROS), which is used to program collaborative robots, such as the Baxter robot from Rethink Robotics.

I. INTRODUCTION

For the better part of the last few decades, robots were successfully deployed to take over tasks from humans. This success has largely been reached because these tasks require skills at which robots excel, such as precise and frequent repetition of tasks that are heavy or difficult to do by humans. However, within the manufacturing industry fully automating more tasks done by humans is no longer possible as "fully automated machines don't evolve on their own" [1]. This increased the interest in human-robot collaboration from within the manufacturing industry. Collaborative robots seem better suited to perform these tasks.

Several research results indicate that people may accept robots as a co-worker or even a coordinator of work [2]. Successful introduction into production facilities however requires collaboration of all stakeholders. Limited involvement of all these stakeholders may lead to frustration with people actually collaborating with the robot [3]. Not all of which may have the technical expertise to understand the low-level information to realize the human-robot interaction and collaboration. One additional level of complexity is that multimodal interaction may be preferred over operation through direct manipulation and buttons [2], [4].

Multimodal interaction is defined by Oviatt [5] as the combination of two or more input modalities (e.g. hand gestures and speech) in a coordinated manner. Difficulties in implementing such type of interaction have been addressed in the human-computing interaction community for many years, resulting in several proposals that use domain-specific languages to describe *executable interaction specifications*. One can use these languages to specify combinations of events representing the coordinated user actions. These actions can be in sequence or in parallel through multiple modalities. These approaches however have largely focused on traditional human-computer interaction, not on human-robot interaction.

In this paper, we discuss how one of these approaches, Hasselt UIMS [6], may be used to define the interaction at several layers of abstraction to ease involvement of the various stakeholders in the discussion of executable interaction specifications. This involvement is facilitated through the multiple levels of abstractions and the flexibility offered by Hasselt UIMS:

- The multiple levels of abstraction allow a compact overview of the overall interaction at the highest level. More details on how this interaction is accomplished are available at a lower level. At the lowest level custom definition of events, including the associated data, permit as detailed control as required.
- The flexibility of reusing groups of composite events in Hasselt UIMS allows programmers to temporarily replace certain input devices or algorithms or even the actual robot, thus facilitating Wizard-of-Oz setups.
- At the lower levels of abstraction, a natural mapping can be realized to the message-based programming paradigm used by the Robot Operating System (ROS) [7], while still keeping the core of Hasselt UIMS independent of ROS.

To reach these goals, Hasselt UIMS was internally redesigned to allow flexible handling of input modalities and reuse of event definitions across projects. The domain-specific language, Hasselt, was extended to allow the natural mapping between the event-based paradigm used by Hasselt and the message-based paradigm used by ROS.

II. RELATED WORK

For more than a decade, a series of languages has been proposed with the intention of describing multimodal interactions in a declarative fashion. Some of these languages

* This research received funding through the Claxon project. ClaXon is a project co-funded by iMinds, a digital research institute founded by the Flemish Government. Project partners are Softkinetic, AMS, Robovision, Audi, Melexis, iMinds-EDM, iMinds-SMIT, R&MM, with project support from IWT and Innoviris.

[1]Jan Van den Bergh, Fredy Cuenca, Kris Luyten and Karin Coninx are with Expertise Centre for Digital Media, UHasselt - tUL - iMinds, Wetenschapspark 2, 3590 Diepenbeek, Belgium {jan.vandenbergh, kris.luyten, karin.coninx}@uhasselt.be

[2]Fredy Cuenca Lucero was with Expertise Centre for Digital Media, UHasselt - tUL - iMinds, Wetenschapspark 2, 3590 Diepenbeek, Belgium fqnk@yahoo.com

describe the data flow from the peripherals to the application; others describe the relations between user events and event handling functions. Both our proposed Hasselt and the languages described below, fit in the latter group. For a more complete sample of multimodal interaction description languages, readers can refer to surveys [8], [9].

With MEngine [10], the sequences of related user events involved in multimodal interactions are depicted as finite state machines (FSMs). Each link represents a user event; the nodes can be labeled with handling functions that are to be launched during the interaction. One issue of this approach is that the FSMs have to be manually specified. Besides, these FSMs are not hierarchical —this compels to describe the human-machine interaction at a single level of abstraction, which may be inappropriate for medium-large size systems.

NiMMiT [11] also allows FSM-based multimodal interaction specifications. Unlike MEngine, each link can represent not only one single user action but also the co-occurrence of several actions. When running NiMMiT models, several handling functions can be fired sequentially in response to a system state transition.

The visual models of HephaisTK [12] have more expressiveness than NiMMiT models. Each link of these FSM-based models can be annotated by a combination of events (e.g. events can be singleton, complementary, redundant, or equivalent). A handling function can be launched when this combination of user events is (automatically) detected by the HephaisTK's framework.

When using ICO [13], multimodal interactions must be described as Petri nets. The transitions of these Petri nets represent user events whereas their places are temporary sites through which the input data flows. The transitions can be annotated to transform the input data and/or to launch handling functions. The high learning curve required to use ICO limits its economical applicability to the field of safety-critical systems [13].

Instead of providing a visual language, Mudra [14] allows textual CLIPS-based specifications. The event sequences that must be handled by the intended multimodal system are defined as a set of rules that the CLIPS engine constantly checks. The satisfaction of a rule leads to the activation of externally defined handling functions.

As with HephaisTK and Mudra, our language allows binding user-defined combinations of events to handling functions. Unlike HephaisTK, developers can edit textual specifications for interactions, while graphical representations are generated that can facilitate discussions within a multi-disciplinary team. Mudra, in contrast, uses rules on a fact database to trigger handling functions. It does not feature a visual notation that can ease communication with non-programmers.

State machines have also been used within the robotics domain [15], [16], [17]. ActionLib [18] is C$\hat{+}$+ project on top of ROS that uses state machines to ease development of longer-lasting preemptable tasks. For simple common usage a single class can be used that hides the state machine logic. These approaches mostly address the development complexity of such systems for programmers.

III. HASSELT AND HASSELT UIMS

Hasselt provides notations that enable programmers to combine several user events into a single abstraction called composite event. These user events may originate from different modalities such as mouse, keys, touch screens, speech or depth cameras. Each composite event can be bound to one or more handling functions. At runtime, the handling functions will be called at different moments of the interaction, in response to the (partial) detection of composite events.

Hasselt UIMS is the supporting tool of Hasselt. It includes the editors, compilers, runtime environment, and debugging tools with which one can describe, run, and trace the execution of Hasselt specifications. Hasselt UIMS uses projects to support configuration of the environment, such as specification of the required external applications and libraries. These projects can also be reused in other projects enabling reuse of sets of composite events (and the related libraries and applications).

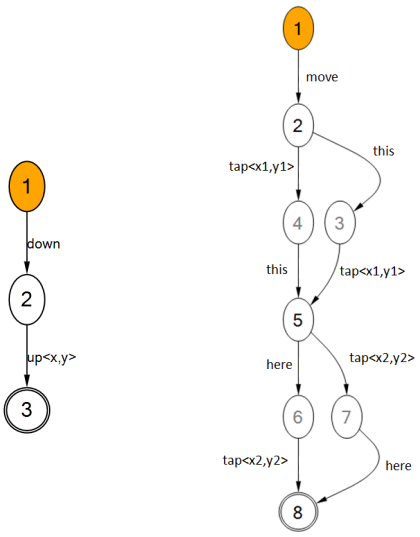
A. Definition of a single multimodal command

We illustrate the definition of a multimodal command with an example that stems from the original office and home environment for which Hasselt UIMS was developed; a variation of the classical put-that-there example for multimodal interaction by Bolt [19]. The user-defined speech-and-touch event *putThatThere* (Equation 1) combines speech and touch events to move an object by linking these events to the methods HIGHLIGHTSELECTEDPHOTO( ) and MOVEHIGHLIGHTEDPHOTO( ) (Equation 2). Both the code for defining composite events and the code for event binding are written with Hasselt. The handling functions were written with C#, a general-purpose language aimed at the .Net platform.

$$\begin{aligned} &event\ \ tap\langle x,y\rangle \ \ = \ \ touch.down;\\ &\qquad\qquad\qquad\qquad\ \ touch.up\langle x,y\rangle \\ &event\ putThatThere = speech.move;\\ &\qquad\qquad\qquad\qquad\ \ speech.that + ce.tap\langle x1,y1\rangle;\\ &\qquad\qquad\qquad\qquad\ \ speech.there + ce.tap\langle x2,y2\rangle \end{aligned} \tag{1}$$

Event names consist of two parts separated by a dot. The first part indicates the modality (speech or touch in Equation 1) or the fact that the event is a combination of other events (*ce*). The second part the event. For the speech modality, an abbreviated form was used for the recognition of simple words; in this case the recognized words are used as event names. For the touch modality, *down* and *up* events are used. Events can (optionally) have different sets of parameters, specified between chevrons.

Composite events, such as putThatThere, are formed by specifying how events can be combined over time using four symbols: The operator $FOLLOWED\ BY(;)$ is used to connect events that occur sequentially; $AND(+)$ is intended to connect events that co-occur in time; $OR(|)$ must be used to connect equivalent events; and the operator $ITERATION(^*)$ serves to describe arbitrarily long sequences of events [6].



(a) Event tap (b) Event putThatThere

Fig. 1: Two FSMs are automatically generated from composite events declared in Equation 1 to represent all relevant interaction states. Different system responses (e.g. calling handling functions, maintaining variables) can be conveyed at each interaction state, as declared in Equation 2.

Usage of $AND(+)$ in a composite event is translated in recognition of the events in any order. For the *putThatThere* event this means that both the sequence with first a tap and than a speech-command (*here* or *there*), and the reverse sequence are encoded in the FSM.

$$
\begin{aligned}
&wrt \;\; ce.tap\langle x,y\rangle \\
&\quad @link(1,touch.down)\ do \\
&\quad\quad assign: t0 = Now.TotalMilliSeconds; \\
&\quad @link(2,touch.up\langle x,y\rangle)\ do \\
&\quad\quad assign: t1 = Now.TotalMilliSeconds; \\
&triggers\ when\ t1 - t0 <= 200
\end{aligned}
\tag{2}
$$

$$
\begin{aligned}
&wrt \;\; ce.putThatThere \\
&\quad @node(5)\ do \\
&\quad\quad call: HighlightSelectedPhoto(x1,y1); \\
&\quad @node(8)\ do \\
&\quad\quad call: MoveHighlightedPhoto(x1,y1,x2,y2);
\end{aligned}
$$

Each composite event is automatically converted into a finite state machine (FSM). The FSMs generated from Equation 1 are shown in Fig. 1. The nodes of these FSMs represent steps of the human-machine interaction. Hasselt users can refer to these nodes in order to indicate the moments when the handling functions are to be called. Hasselt also allows specifying (temporal and/or spatial) constraints among the elementary events comprising a composite event. E.g. Equation 2 specifies that to detect a tap, the time between touch down and touch up should be at most 200 milliseconds.

The code also shows how to bind several handling functions to a single composite event: HIGHLIGHTSELECTEDPHOTO( ) and MOVEHIGHLIGHTEDPHOTO( ) will be called right after object selection and when the event *putThatThere* is fully detected, both of these moments are represented as *node*(5) and *node*(8), respectively, in the FSMs shown in Figure 1.

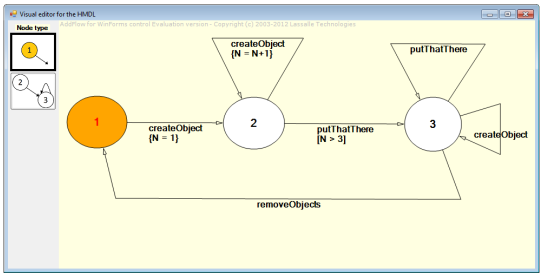


Fig. 2: Human-machine dialog using the composite events declared in Equations 1, 2. The number of created objects (N) puts additional constraints on which actions are possible at a specific moment.

B. Definition of human-machine dialog

One can directly use finite state machines (FSMs) to describe human-machine dialogs (HMDs) using the composite events discussed above. HMDs are useful to describe systems whose responses vary depending on the current operation mode, e.g. a touch on the arm robot may be an indication for the robot to start working, but the same event may be ignored once the robot is working. Hasselt dialog models show the operation modes (e.g. idle/working) of the intended system and all the interactions available in each operation mode. FSMs are visual specifications that emphasize flow. Such notations are useful to discuss processes in interdisciplinary teams [20].

The HMDs elaborated with Hasselt look like Fig. 2. Each node represents a different state. Transitions between nodes can be triggered by composite events. These transitions can also be subject to conditions that reflect the system state (expressed between square brackets); in this case the composite event putThatThere can only be executed once at least three objects are created. After removal of all objects, the system returns to its begin state.

To better understand Hasselt UIMS workflow, readers can refer to a publicly available video[1] that shows how to create and run a prototype that involves composite events, a HMD and an .Net application.

IV. SUPPORT ITERATIVE DEVELOPMENT OF HUMAN-ROBOT COLLABORATION

The previous section discussed Hasselt and how to specify composite events. These events can be reused in other composite events as can be seen in Equation 1, where the event *putThatThere* reuses the previously defined event *tap*<*x,y*>. The event *putThatThere* is used to make a translation from the problem domain, move this [object] here, to events in the solution domain, taps. These taps are defined in terms of events that are generated by an input recognizer, an event

[1]Hasselt UIMS workflow:
`https://www.youtube.com/watch?v=jC5EuBYWWRc`

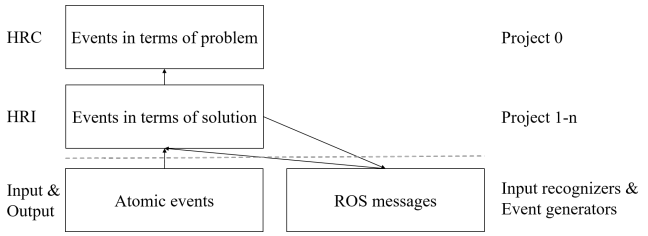


Fig. 3: Hasselt supports events at different levels of abstraction. Events above the dashed line are defined in one or more Hasselt projects, events and messages below it are defined in external libraries by extending the classes in Fig. 6.

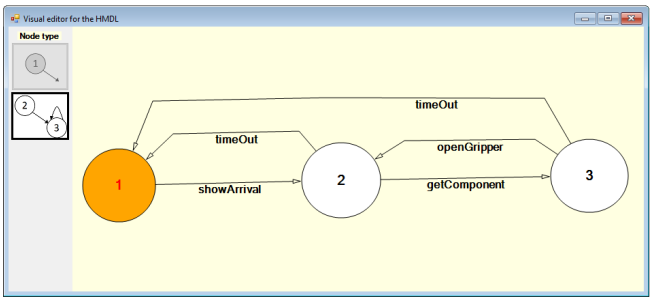


Fig. 4: Hasselt notation for a simple human-robot collaboration scenario in which a robot can give components to the worker on request.

source such as a touch controller, that generates *atomic events*.

A. Hasselt UIMS for prototyping HRC

This ability to hierarchically compose events can be used to one's advantage during the development of human-robot collaboration (and other applications); once a proposal for a scenario is created, composite events can have names in the problem domain.

The composite events *showArrival*, *getComponent*, *openGripper*, and *timeOut* were defined for a simple scenario in which specific workers can ask the Baxter collaborative robot to give a component to them and to release it once they are ready to take the component. Fig. 4 shows an HMD that gives an graphical overview of which actions will be taken to complete the scenario using these events. This visualization was effectively understood during discussions in our multi-disciplinary team and provided a direct starting point for discussion of the specific interactions in the scenario.

When one defines this scenario in a separate project from the event definitions, varying the projects that are included in the HMD's project allows experimentation with how these actions can be realized. One of these projects can contain composite events that map the composite events to mouse and keyboard (or touch) actions to explore variations of the HMD without dependencies on any specific soft- or hardware that may be necessary to realize the final scenario. One can also make partial links to the collaborative robot to facilitate

TABLE I: Mapping between concepts in ROS and in Hasselt

Message-based (ROS)	Event-based (Hasselt)
topic	input recognizer
message type	event name
fields	parameters

Wizard-of-Oz testing [21], in which certain human input is recognized by a human instead of e.g. an artificial intelligence component to gather requirements of how humans would give certain input without being constrained to what is technically possible at that moment in time in that specific project.

B. Technical updates to Hasselt UIMS

To realize this, we updated Hasselt UIMS [6] so that it no longer requires all input recognizers to be built-in. Instead these input recognizers can now be loaded from libraries. Together with the fact that one could already call functions from external libraries this makes it possible to get input from and generate actions to be performed by collaborative robot when desired. When no robot is available nor software to communicate with itanother project that does not depend on the robot software can be used. This approach is illustrated in Fig. 3; one project is used to define human-robot collaboration (the overall HMD), different projects can be used to define how each interaction, defined using one or more composite events, which ultimately use atomic events defined in separate libraries.

In a situation where the collaborative robot can be programmed using the Robot Operating System [7] (ROS) another approach to do communication with other software components seems appropriate. ROS uses a message-based programming model in which (small) programs (nodes) communicate by sending messages to topics and receiving messages from topics to which they are subscribed. We can thus exploit the similarity between event-based and message-based programming to create a mapping as can be seen in Table I.

This mapping allows hiding some boilerplate code from the user of Hasselt UIMS; handling subscriptions is done in the library and ROS messages can be handled in the same way as events from any other input recognizer. The call functionality, as used in Equation 2 to trigger functionality defined in external libraries, however does not fit well in this mapping.

In order to make the mapping complete, we introduced the possibility to *raise* events. These can be *hasselt* events, which are only used within Hasselt UIMS, as well as external events for which an external library defines what happens. The syntax to raise events combines the syntax for calling a function and that of catching an atomic event: *raise: "eventGenerator"."eventName"<"event parameters">;*.

Equation 3, Equation 4 and Fig. 5 illustrate the use of this syntax using a example where the *openGripper* event is triggered when a *thumbUp* gesture for at least half a second. The worker collaborating with the robot can make

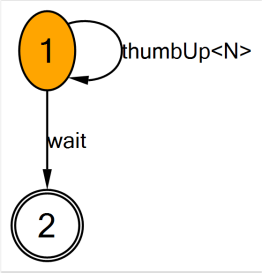


Fig. 5: Finite state machine generated from composite event openGripper in Equation 3, and annotated by Equation 4

this gesture with the non-dominant hand, while the dominant hand is ready to take a component from the gripper.

$$\begin{array}{l} event \;\; wait = delay - 1000; \\ \\ event \;\; openGripper = skHands.thumbUp < N > *; ce.wait \end{array} \tag{3}$$

$$\begin{array}{l} wrt \;\; ce.openGripper \\ \quad @node(0) \;\; do \\ \quad\quad assign : thumbCounted = 0; \\ \quad @link(1, skHands.thumbUp\langle N \rangle) \;\; do \\ \quad\quad assign : thumbCounted = thumbCounted + 1; \\ \quad @node(2) \;\; do \\ \quad\quad assign : thumbCounted = 0; \\ \quad @link(1, ce.wait) \;\; do \\ \quad\quad raise : inputAction.string <' openGripper' >; \\ \quad\quad speak :' OK'; \\ \quad when \;\; thumbCounted > 5; \end{array} \tag{4}$$

We now explain the code in Equation 4 into more detail: the *skHands* input recognizer, can detect a *thumbUp* event every 100 milliseconds. The finite state machine in Figure 5 generated from the *openGripper* event, to count the number of thumbUp events during 1 second as defined in the wait event. The frequency of updates is exploited in Equation 4 to express the number of expected correctly detected thumbUp events in this period. As the gesture recognition in not perfect, the treshold of more than 5 thumbUp events should be exceeded before a *std_msgs/String* message is sent to the topic */input/action*. To support prototyping without the actual robot present, a spoken message is given in addition to sending the message.

As the example illustrates, we opted to not directly use the topic names within Hasselt, but to transform it to the naming convention already in use for input recognizers within Hasselt UIMS. Similarly, the name of the message type is abbreviated. The way we realized the system, however, means that these choices can be changed if circumstances would dictate so. In a situation where all team members involved in the definition of these composite events are familiar with ROS, keeping the naming scheme closer to the ROS naming scheme can be considered as the mapping is realized by implementing an interface of Hasselt UIMS in a separate library.

The interfaces / classes that should be implemented to allow detection of atomic events or generation of events / publishing of ROS messages are shown in Fig. 6. A

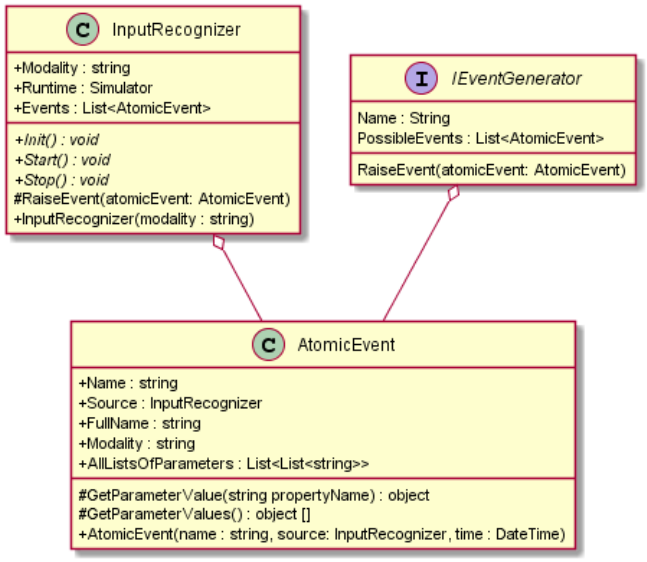


Fig. 6: Interface and classes that should be implemented by input recognizers (subscribed topics in ROS) or event generators (topics to publish on in ROS) for Hasselt UIMS.

library should declare the events it can detect or generate, including some information to support editing in Hasselt UIMS. Properties in the event classes correspond to event parameters or message fields.

Hasselt UIMS, together with the libraries it uses, can be considered as a single node within ROS. Within Hasselt UIMS the message-based communication can be programmed using (compositions of) events. Its user can define when certain events (messages) should be detected and how the "Hasselt UIMS node" should react to them, including raising new events (messages).

V. DISCUSSION

This paper presented how multiple levels abstractions can be used to create executable specifications of multimodal human-robot collaboration. Finite state machines are used to define the overall human-robot dialog. This dialog, expressed using finite state machines, can gradually and flexibly be refined using composite events that are specified using a combination of a textual domain-specific language and automatically generated finite state machines that provide a detailed graphical overview of each specific human-robot interaction.

One of the strengths of this approach is that it allows to create abstractions that divert from the technicalities of robot programming so that the (high-level) code and diagrams become more accessible to people less familiar with robotics programming. This way we want to enable people that have limited technical knowledge to make limited updates to allow more flexibility in fine-tuning the interaction. We however realize that the interaction specification can still be made more accessible, especially for people with less programming expertise. We are currently evaluating some limited adaptations to these notations with people from

different disciplines active in the human-robot interaction domain.

The focus of our contribution is thus different from earlier approaches that aimed to facilitate iterative development [22], [23], as these approaches focused solely on the implementer of the robotic system, not on supporting the overall team. Flexibility in the instantiation of interaction modalities as offered by Hasselt UIMS can be important to be able to prototype interaction in a flexible manner.

We are currently using Hasselt UIMS in the realization of human-robot collaboration scenarios that are more complex than the one discussed in section IV-A and Fig. 4. These scenarios combine multiple input modalities and feedback channels by the robot including screen and motion feedback, with a multi-disciplinary team. We are using Hasselt to refine the overall scenario and define and execute the human-robot interactions with Baxter, a co-bot from Rethink Robotics. At a later stage interactions with a more traditional robot will be specified. Hasselt UIMS is and will be used in combination with ROS to define the human-robot interaction.

The current implementation requires manual coding of the libraries that interact with specific ROS topics. Which means that additional coding is still required to add support for additional messages and/or topics. This is an area that can be addressed in future work.

We are evaluating Hasselt with representatives from multiple-disciplines involved in research and practice of human-robot interaction in a manufacturing setting. Another area of future work is to investigate how the transition from an approach that focuses on rapid prototyping to an approach that can be sustainably deployed.

REFERENCES

[1] C. Trudell, Y. Hagiwara, and M. Jie, "Humans replacing robots herald Toyota's vision of future," Apr. 2014. [Online]. Available: http://www.bloomberg.com/news/articles/2014-04-06/humans-replacing-robots-herald-toyota-s-vision-of-future

[2] M. C. Gombolay, R. A. Gutierrez, S. G. Clarke, G. F. Sturla, and J. A. Shah, "Decision-making authority, team efficiency and human worker satisfaction in mixed human–robot teams," *Autonomous Robots*, vol. 39, no. 3, pp. 293–312, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10514-015-9457-9

[3] D. Wurhofer, T. Meneweger, V. Fuchsberger, and M. Tscheligi, "Deploying robots in a production environment: A study on temporal transitions of workers' experiences," in *Human-Computer Interaction – INTERACT 2015: 15th IFIP TC 13 International Conference, Bamberg, Germany, September 14-18, 2015, Proceedings, Part III*, J. Abascal, S. Barbosa, M. Fetter, T. Gross, P. Palanque, and M. Winckler, Eds. Cham: Springer International Publishing, 2015, pp. 203–220. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-22698-9_14

[4] G. Randelli, T. M. Bonanni, L. Iocchi, and D. Nardi, "Knowledge acquisition through human–robot multimodal interaction," *Intelligent Service Robotics*, vol. 6, no. 1, pp. 19–31, 2012. [Online]. Available: http://dx.doi.org/10.1007/s11370-012-0123-1

[5] S. Oviatt, in *The Human-computer Interaction Handbook*, J. A. Jacko and A. Sears, Eds. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 2003, ch. Multimodal Interfaces, pp. 286–304. [Online]. Available: http://dl.acm.org/citation.cfm?id=772072.772093

[6] F. Cuenca, J. Van den Bergh, K. Luyten, and K. Coninx, "Hasselt uims: A tool for describing multimodal interactions with composite events," in *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ser. EICS '15. New York, NY, USA: ACM, 2015, pp. 226–229. [Online]. Available: http://doi.acm.org/10.1145/2774225.2775437

[7] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[8] F. Cuenca, K. Coninx, K. Luyten, and D. Vanacken, "Graphical Toolkits for Rapid Prototyping of Multimodal Systems: A Survey," *Interacting with Computers*, vol. 27, no. 4, pp. 470–488, 2015. [Online]. Available: http://dx.doi.org/10.1093/iwc/iwu003

[9] B. Dumas, D. Lalanne, and S. Oviatt, "Multimodal Interfaces: A Survey of Principles, Models and Frameworks," *Human Machine Interaction*, vol. 5440, pp. 3–26, 2009.

[10] M.-L. Bourguet, "A toolkit for creating and testing multimodal interface designs," in *Proc. of UIST'02*, 2002, pp. 29–30.

[11] J. De Boeck, D. Vanacken, C. Raymaekers, and K. Coninx, "High level modeling of multimodal interaction techniques using NiMMiT," *Journal of Virtual Reality and Broadcasting*, vol. 4, no. 2, 2007.

[12] B. Dumas, D. Lalanne, and R. Ingold, "Description Languages for Multimodal Interaction: A Set of Guidelines and its Illustration with SMUIML," *Journal of multimodal user interfaces*, vol. 3, no. 3, pp. 237–247, 2010.

[13] D. Navarre, P. Palanque, J.-F. Ladry, and E. Barboni, "Icos: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability," *ACM Trans. Comput.-Hum. Interact.*, vol. 16, no. 4, pp. 18:1–18:56, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1614390.1614393

[14] L. Hoste, B. Dumas, and B. Signer, "Mudra: A unified multimodal interaction framework," in *Proceedings of the 13th International Conference on Multimodal Interfaces*, ser. ICMI '11. New York, NY, USA: ACM, 2011, pp. 97–104. [Online]. Available: http://doi.acm.org/10.1145/2070481.2070500

[15] I. Lütkebohle, R. Philippsen, V. Pradeep, E. Marder-Eppstein, and S. Wachsmuth, "Generic middleware support for coordinating robot software components: The task-state-pattern," *Journal of Software Engineering for Robotics*, vol. 2, no. 1, pp. 20–39, 2011.

[16] M. Loetzsch, M. Risler, and M. Jungel, "Xabsl-a pragmatic approach to behavior engineering," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 5124–5129. [Online]. Available: http://dx.doi.org/10.1109/IROS.2006.282605

[17] J. Fabry and M. Campusano, *Advances in Artificial Intelligence – IBERAMIA 2014: 14th Ibero-American Conference on AI, Santiago de Chile, Chile, November 24-27, 2014, Proceedings*. Cham: Springer International Publishing, 2014, ch. Live Robot Programming, pp. 445–456. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-12027-0_36

[18] "actionlib package summary (wiki)," 12 2015, last accessed on May 23, 2016. [Online]. Available: http://www.ros.org/wiki/actionlib

[19] R. A. Bolt, ""Put-that-there": Voice and Gesture at the Graphics Interface," *SIGGRAPH Comput. Graph.*, vol. 14, no. 3, pp. 262–270, July 1980. [Online]. Available: http://doi.acm.org/10.1145/965105.807503

[20] K. Li, A. Tiwari, J. Alcock, and P. Bermell-Garcia, "Categorisation of visualisation methods to support the design of human-computer interaction systems," *Applied Ergonomics*, vol. 55, pp. 85 – 107, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0003687016300096

[21] N. Dahlbäck, A. Jönsson, and L. Ahrenberg, "Wizard of oz studies: Why and how," in *Proceedings of the 1st International Conference on Intelligent User Interfaces*, ser. IUI '93. New York, NY, USA: ACM, 1993, pp. 193–200. [Online]. Available: http://doi.acm.org/10.1145/169891.169968

[22] P. Estefó, M. Campusano, L. Fabresse, J. Fabry, J. Laval, and N. Bouraqad, "Towards live programming in ros with pharos and lrp," *arXiv preprint arXiv:1412.4629*, 2014.

[23] S. Adam and U. P. Schultz, "Towards interactive, incremental programming of ros nodes," *arXiv preprint arXiv:1412.4714*, 2014.