Challenges in Synchronized Behavior Realization for Different Robotic Embodiments

Iwan de Kok, Jacqueline Hemminghaus, and Stefan Kopp

Social Cognitive Systems, CITEC, Bielefeld University
idekok@techfak.uni-bielefeld.de

Abstract. Humanoid robots are envisioned to engage in natural and fluid interaction with human interaction partners. In such a natural interaction situation, highly adaptive planning and execution of multimodal behavior is required. Here, we present work using a behavior realizer that was build for virtual humans and apply this on two robot platforms with different embodiments, Nao and Furhat. This architecture combines planning-based and event-based synchronization techniques that allow changing behavior even at execution time.

Keywords: Multimodal Behavior Generation, Continuous Interaction, Human-Robot-Interaction, Social Robotics, AsapRealizer, BML

1 Introduction

Humanoid robots are envisioned to engage in natural and fluid interactions with their human interaction partners in rich environmental settings. For this, they need to be able to combine speech with nonverbal behaviors like gaze, head or hand gestures in synchronous multimodal utterances. Such utterances need to convey the robot's communicative goals and must be adaptive to a changing environment and conversation structure. Importantly, multimodal utterances are not first planned and then simply executed. Instead, communicative behavior is produced incrementally in chunks, with on-the-fly adaptation to running and succeeding increments.

For social and/or humanoid robots, generating such fluent and convincing multimodal communicative behavior has been a great challenge. Different robot platforms come with different embodiments that have different communicative affordances. Furthermore, they have inherent limitations, e.g., with regard to range, continuity, accuracy or predictability of their motions. These issues have been a focus of attention in the virtual agents community for some time.

Here, we present the current state of our integration of previous work on virtual agents into our work with two robot platforms with different embodiments, Nao and Furhat, and the obstacles that arise in this effort. In Section 2 we will present the key extensions to BML we aim to use with our robot platforms. Section 3 explains how the behavior specified in BML is realized for virtual agents using engines. In Section 4 we present the implementation of two engines for Nao and Furhat and discuss the obstacles we face.

2 Extended Behavior Markup Language

Behavior Markup Language (BML) [4] is an XML based language that is used in the virtual agents community to specify the multimodal behavior the behavior realizer needs to perform on the agent. A main feature of BML is the ability to specify the synchronization constraints between behaviors within a BML block, such as between speech, gestures and gaze. Each behavior is divided in up to six animation phases. Each time point at a phase boundary is a so-called sync-point. The seven sync-points are: *start*, *ready*, *stroke-start*, *stroke*, *stroke-end*, *relax* and *end*. These sync-points can be used to synchronize behaviors, i.e., by assigning the start of behavior A to the stroke of behavior B.

In previous work, BML was extended with several features enabling a continuous interaction. These extensions, called BMLA, mainly target the ability to interact with and synchronize between already planned BML blocks. These extensions are the following:

- **Preplanning** - Plan and synchronize all behaviors within the BML block, but wait for a signal before execution. [2]
- **Interrupt** - Cancel the specified BML block or specified behaviors [2].
- **OnStart** - Start the BML block at the same time as another (preplanned) BML block [1].
- **AppendAfter** - Attach the new BML block A after the specified BML block B (viz. start of the first behavior in A is end of last behavior in B) [5].
- **ChunkAfter** - Attach the new BML block A after the specified BML block B, but skipping the retraction phase of the last behavior in BML block B if possible (viz. start of the first behavior in A is relax of last behavior in B) [5].

These extensions enable the behavior planner to change and attach to already planned behavior enabling continuous interaction.

Besides specifying communication from behavior planner to the behavior realizer BML also specifies feedback messages the realizer sends back to the planner keeping it up-to-date on the life cycle of a BML block. These feedback message are *Prediction Feedback*, which provides information about the expected realization of a BML block, *Progress Feedback*, which provides real-time information about the progress of an ongoing BML Block and its sync-points, and *Warning Feedback*, which informs the realizer that a BML block or parts of it have failed to realize.

The feedback messages include local scheduled timing information of the individual behaviors. This keeps the behavior planner up-to-date and enables it to continuously monitor the agent's behavior and adjust it if needed.

3 Behavior Realization with Engines

In AsapRealizer, Engines are modules specialized in planning and executing the corresponding motor behaviors for a specific modality, e.g. speech, facial and

body animation, including capabilities of predicting behavior timings, monitoring and controlling a behavior. With the prediction component the Engine is capable of making assumptions about the timings of a behavior, e.g., its duration or end point as well as timings of known sync-points relative to the start time. The monitoring module allows the Engine to keep track of the current state of a behavior. It informs the Engine about reaching a specific intermediate sync-point or at least about the start and end and the corresponding timing. Finally, Engines have Motor Control Units that translate behaviors into agent specific controlling commands.

Predictions are required for planning-based scheduling, while monitoring is required for event-based synchronization. When planning a motor behavior for a virtual agent, the corresponding Engine starts to predict the timings of the behavior first. To predict the timings, e.g. of a gesture phase, the Engine uses an internal forward model that allows to compute when a specific joint reaches a predefined angle or position. For speech, the connected speech synthesis provides the Engine with necessary timing information, such as the duration of the speech, timings of marked words and phoneme timings.

Monitoring, in general, is not necessary when controlling virtual agents because they are only interacting in a simulated environment without constraints interfering with the preplanned joint movements. That is, unless collision detection is in the environment, which typically is not done. The same applies to monitoring speech. In this case, speech's synchronization timings and duration, already evaluated in the prediction process, do not change during synthesis. If further extensions require monitoring of the agents behavior, skeleton and joint information are easily requested from the internal model and speech timings from the underlying synthesis.

However, when controlling a robot, one is limited to the functionalities the robot platform (including its API) provides. Most robots provide information through events generated by the robot, but these events mostly inform about the end of a behavior but not about the actual duration of it. This raises the question how the timing of a behavior can be predicted if the used robot's API does not provide this information. One possible way of predicting the time needed to perform, e.g., a gesture is calculating the inverse kinematics for the gesture and applying that to an internal model and use the time needed as predicted gesture duration. Some robots are already accompanied by a simulation, e.g., Nao with a simulator[1] or Furhat with Iristk [3] that could be used as internal model. But their main task is to visualize how a behavior might look like in the real-world, while ignoring constraints, e.g., physics, motor speed, collisions or joint limits, that might influence the gesture execution. Additionally using these simulations to predict time constraints of a behavior to create and predict motor plans can get time consuming because they normally run in real-time and are not adaptable regarding their execution time.

In consequence, for robots monitoring gets more important in order to keep the timing of a behavior up-to-date and to ensure multimodal synchrony.

[1] https://www.cyberbotics.com

4 Use Case: Nao and Furhat

We aim to transfer the behavior generation and realization processes that are already well explored in the field of virtual agents, including gesture alignment and flexible gesturing as well as context adaptation, to the field of human-robot interaction. Because of this we extended AsapRealizer with two robot-specific Engines: one for Nao and one for Furhat.

The challenge with both robots is that they have different embodiments with different capabilities to communicate, e.g. the Nao could point to some object where the talking head Furhat could only gaze at them. Additionally both robots are controlled through different APIs. Furhat is controlled with Iristk [3] that provides only a small amount of high-level action commands without the possibility of directly controlling joints or blend shapes. In contrast, NaoQi[2] provides functionalities to send high-level action commands, e.g. predefined gestures, as well as functionalities to control and manipulate joint angles directly as well as interpolating between them providing a smooth movement.

Prediction - Apart from different ways of controlling the robots, they both do not provide information about the duration of speech or the execution time of a gesture that could be used as prediction feedback. This information is necessary for planning-based scheduling, e.g., to align a gesture with an important word in speech. Depending on the speed of the actuators, the agent needs to start some time before this important word in order to be ready to execute the gesture on time. That is, when an agent tries to move the attention of its interaction partner to a specific point, he might say "Look at this!". Now the interaction partner does not know exactly where to look, unless the agent uses a pointing gesture to highlight the exact position. In this case it is important that the pointing gesture has reached its final position when the agent says "this" so that the interaction partner can resolve the reference correctly. In order to align spoken words and gestures correctly, it is important to know at which time point a word is spoken before starting the actual speech such that the gesture can be prepared. Therefore, as usual, the speech is preplanned for both robots using Windows' speech synthesis, which provides us with information about the duration of the speech and allows for synchronizing timing and position of marked synchronization points.

Furthermore we add predicted durations for gesture phases corresponding to the execution speed of the robot. This enables the Engine to make assumptions about the duration of a gesture if no gesture phases are explicitly defined in the BML block as well as to provide enough time for the robot to execute the behavior safely. Additionally with the predicted durations of a gesture phase the Engine is able to preplan a gesture to reach a specific synch-point in time. Currently, predictions are made heuristically based on measurements of the robots' kinematics. It is well conceivable to employ a more sophisticated forward model that the robot can even learn itself using a form of body babbling.

[2] http://doc.aldebaran.com/2-4/naoqi/index.html

Chunking and Appending - With the ability of predicting the duration of gesture phases, chunking and appending behaviors (BML blocks, as it were) become possible even when controlling the robots with their limited APIs. Furthermore gesture definitions should account for the six gesture phases or, at least, the ready, stroke and relax sync-points to exploit the capabilities of BMLA. For simplicity and to focus more on the general applicability of these extensions, complex gestures, e.g. pointing and waving, are defined without explicit gesture phases.

Interruption - Behaviors need to be adaptable while executing them, e.g., increasing the volume of the voice while speaking, or interruptible, e.g., stopping to speak because the interaction partner started talking again in order to hold the floor. We first focus on the interruption of already running behaviors as this depends heavily on the functionalities the robots APIs provide.

Usually interrupting a running behavior means instantly stopping the movement or speech. This could be weakened by a graceful interrupt that interrupts the behavior and lets the agent go back into its resting pose. For both robots interrupting speech is already available. For Nao it is also possible to interrupt predefined and named behaviors but for dynamically defined behaviors there is no function available. In order to solve the problem of interrupting a dynamic gesture behavior, we employed the following solutions afforded by NaoQi and our implementation.

- Calling an interrupt function for named behaviors.
- Interrupting the Thread executing the interpolation function. This function has to be executing in a separate Thread, because interpolation is a blocking function call.
- Applying the resting pose to Nao.
- A combination of the second and third options.

In case of Furhat, its gestures are not intended to be interruptible at all, which could be explained by the type of gestures Furhat provides, e.g. head gestures and facial expression, which usually cover a small amount of time. Furthermore, they are either controlled by sending a target position, in terms of gaze, or by defining abstract but fully defined movements, e.g., how many frames does the robot have to reach a certain position and go back. These are then interpolated by the API to reach the defined target point, respectively targeted gesture.

5 Conclusion

We have discussed how a state-of-the art framework for multimodal behavior generation in virtual agents could be extended and applied to two robot platforms, Nao and Furhat. Using the APIs provided by these platforms we have connected the behavior realizer to transfer the features of the architecture that allow us to plan, synchronize and monitor the execution of multimodal utterances. We have found that the APIs of these platforms lack several features that are required to achieve fluid multimodal behavior realization.

To be able to accurately plan and synchronize behavior, one needs accurate prediction times for how long it takes to execute gestures, and the word and phoneme timings for speech. Unfortunately, these prediction timings are not all available through the API.

For monitoring and adapting ongoing behavior the APIs are currently lacking in features as well. The APIs currently do not give detailed progress feedback during execution. Feedback from the robots is typically limited to the start and end times of the gestures. The multitude of sync-points of a gesture are absent. Also, the ability to interrupt behaviors, especially gestures, is missing. This limits our ability to go from one gesture fluently into the other. In the example with Nao, we were unable to make a pointing gestures after a waving gesture without returning to the rest pose first.

This shows that achieving the same fluid multimodal behavior realization with humanoid robots as we can with virtual humans is still a challenge. The current generation of robots platforms do not come with a powerful enough API to plan and adapt multimodal utterances.

Acknowledgments

This work was supported by the BabyRobot project supported by the EU Horizon 2020 Program, grant number: 687831, and by the Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277) at Bielefeld University, funded by the German Research Foundation (DFG).

References

1. Kopp, S., van Welbergen, H., Yaghoubzadeh, R., Buschmeier, H. An architecture for fluid real-time conversational agents: Integrating incremental output generation and input processing. Journal on Multimodal User Interfaces. 8, 97-108 (2014).
2. Reidsma, D., Truong, K. P., van Welbergen, H., Neiberg, D., Pammi, S., de Kok, I., and Straalen van, B. Continuous Interaction with a Virtual Human. In: Proceedings of the 6th International Summer Workshop on Multimodal Interfaces (eNTERFACE'10). 24-39 (2010).
3. Skantze, G., and Al Moubayed, S. IrisTK: A Statechart-Based Toolkit for Multi-Part Face-to-Face Interaction. In: Proceedings of the 14th ACM International Conference on Multimodal Interaction. 69-76 (2012).
4. Vilhjálmsson, H., Cantelmo, N., Cassell, J., E. Chafai, N., Kipp, M., Kopp, S., Mancini, M., Marsella, S., Marshall, A. N., Pelachaud, C., Ruttkay, Z., Thórisson, K. R., van Welbergen, H., and Werf, R. J. The Behavior Markup Language: Recent Developments and Challenges. Intelligent Virtual Agents. 99-111 (2007).
5. van Welbergen, H., Baumann, T., Kopp, S., Schlangen, D. Incremental, Adaptive and Interruptive Speech Realization for Fluent Conversation with ECAs. Intelligent Virtual Agents. p. 468-469 (2013).