A Generic Architecture for a Companion Robot*

Bas R. Steunebrink, Nieske L. Vergunst, Christian P. Mol,
Frank P.M. Dignum, Mehdi Dastani, and John-Jules Ch. Meyer

Intelligent Systems Group, Institute of Information and Computing Sciences
Utrecht University, The Netherlands
{bass,nieske,christian,dignum,mehdi,jj}@cs.uu.nl

Abstract. Despite much research on companion robots and affective virtual characters, a comprehensive discussion on a generic architecture is lacking. We compile a list of possible requirements of a companion robot and propose a generic architecture based on this list. We explain this architecture to uncover issues that merit discussion. The architecture can be used as a framework for programming companion robots.

1 Introduction

Recently, research in companion robots and affective virtual characters has been increasing steadily. Companion robots are supposed to exhibit sociable behavior and perform several different kinds of tasks in cooperation with a human user. Typically, they should proactively assist users in everyday tasks and engage in intuitive, expressive, and affective interaction. Moreover, they usually have multiple sensors and actuators that allow for rich communication with the user. Of course, the task of designing and building a companion robot is highly complex.

Companion robots and affective virtual characters have already been built up to quite advanced stages. However, teams wishing to research companion robots often have to start from scratch on the software design part, because it is hard to distill a firm framework from the literature to use as a basis. Of course many figures representing architectures have been published, but it remains difficult to find out how existing companion robots really work internally. This may be due to most publications focusing on test results of the overall behaviors rather than on explaining their architectures in the level of detail required for replication.

The lack of emphasis on architectures may be caused by much of the research on companion robots being driven by the teams' research goals, resulting in their architectures mostly being designed to support just the desired behaviors instead of being generic for companion robots. If there were a good generic architecture for companion robots, a (simple) default implementation could be made, providing (new) researchers with a framework that they could use as a starting point. Depending on the application domain and research goals, some default implementations of modules constituting the architecture may be replaced to achieve the desired custom behavior, while other modules can just be readily used to complete the software of the companion robot.

* This work supported by SenterNovem, Dutch Companion project grant nr: IS053013.

Of course, anyone wishing to build the software of a companion robot can just start up his/her favorite programming environment and try to deal with problems when they occur, but obviously this is not a very good strategy to follow. Instead, designing and discussing an architecture beforehand raises interesting issues and allows questions to be asked that otherwise remain hidden. Indeed, there are many non-trivial choices that have to be made, pertaining to e.g. distribution and assignment of control among processes, synchronization of concurrent processes, which process is to convert what data into what form, where to store data in what form, which process has access to which stored data, which process/data influences which other process and how, the types of action abstractions that can be distinguished (e.g. strategic planning actions, dialogue actions, locomotion actions), the level of action abstraction used for reasoning, who converts abstract actions into control signals, how are conflicts in control signals resolved, what are the properties of a behavior emerging from a chosen wiring of modules, what defines the character/personality of a companion robot (is it stored somewhere, can its parameters be tweaked, or does it emerge from the interactions between the modules?). Answers to these and many other questions may not be obvious when presented with a figure representing an architecture, but these issues can be made explicit by proposing and discussing one.

In this paper we introduce an architecture which is generic for companion robots and explain it in as much detail as possible in this limited space. This architecture contains the components necessary to produce reasonably social behavior given the multimodality of a companion robot's inputs and outputs. We do not claim that the proposed architecture represents the ultimate companion robot architecture. Rather, the aim of this paper is to provoke a discussion on the issues and choices involved in designing the software of a companion robot. Ultimately, this work could lead to the implementation of a generic framework that could be used as a basis for the software of new companion robots.

This paper is outlined as follows. In Section 2, we gather a list of possible requirements for a companion robot and introduce our architecture satisfying these requirements. In Section 3, we treat the *functional components* in the architecture in more detail. In Section 4, we connect the functional components by explaining the *interfaces* between them. We discuss some related work in Section 5 and finish with conclusions and plans for future research in Section 6.

2 Possible Requirements for a Companion Robot

In order to come up with a generic architecture suitable for companion robots, we must first investigate the possible requirements for a companion robot. These requirements are optional, meaning that only the 'ultimate' companion robot would satisfy them all. In practice however, a companion robot does not have to. The actual requirements depend on various factors, such as the application area of the robot and its hardware configuration. However, below we compile a list, as exhaustive as possible, of possible requirements which a generic architecture must take into account.

First of all, a companion robot should be able to perceive the world around it, including auditory, visual, and tactile information. The multimodality of the input creates the need for synchronization (e.g., visual input and simultaneously occurring auditory input are very likely to be related), and any input inconsistent over different modalities should be resolved. Moreover, input processors can be driven by *expectations* from a reasoning system to focus the robot's attention to certain signals. Of course, any incoming data must be checked for relevancy and categorized if it is to be stored (e.g., to keep separate models of the environment, its users, and domain knowledge).

A companion robot should be able to communicate with the user in a reasonably social manner. This means not only producing sensible utterances, but also taking into account basic rules of communication (such as topic consistency). In order to maintain a robust interaction, a companion robot must always be able to keep the conversation going (except of course when the user indicates that he is done with the conversation). This also involves real-time aspects; e.g., to avoid confusing or boring the user, long silences should not occur in a conversation.

Additionally, a companion robot is likely to be designed for certain specific tasks, besides communicating with its users. Depending on e.g. the domain for which the companion robot is designed and the type of robot and the types of tasks involved, this may call for capabilities involving planning, physical actions such as moving around and manipulating objects, or electronic actions (e.g., performing a search on the internet or programming a DVD recorder). Proactiveness on part of the robot is often desirable in tasks involving cooperation.

A companion robot should also exhibit some low-level reactive behaviors that do not (have to) enter the reasoning loop, such as blinking and following the user's face, and fast reactive behaviors such as startling when subjected to a sudden loud noise. To make the interactions more natural and intuitive, a companion robot should also be able to form and exhibit emotions. These emotions can be caused by cognitive-level events, such as plans failing (disappointment), goal achievement (joy), and perceived emotions from the user (if negative: pity). Reactive emotions like startle or disgust can also influence a robot's emotional state. Moreover, emotions can manifest themselves in many different ways; e.g., facial expressions, speech prosody, selecting or abandoning certain plans, etc.

Finally, a companion robot should of course produce coherent and sensible output over all available modalities. Because different processes may produce output concurrently and because a companion robot typically has multiple output modalities, there should be a mechanism to synchronize, prioritize, and/or merge these output signals; e.g., speech should coincide with appropriate lip movements, which should overrule the current facial animation, but only the part that concerns the mouth of the robot (provided it has a mouth with lips).

In Figure 1, we present a generic architecture for companion robots which accounts for the requirements described above. Note that we abstract from specific robot details, making the architecture useful for different types of companion robots. We emphasize again that this is an architecture for an 'ultimate' companion robot; in practice, some modules can be left out or implemented empty.

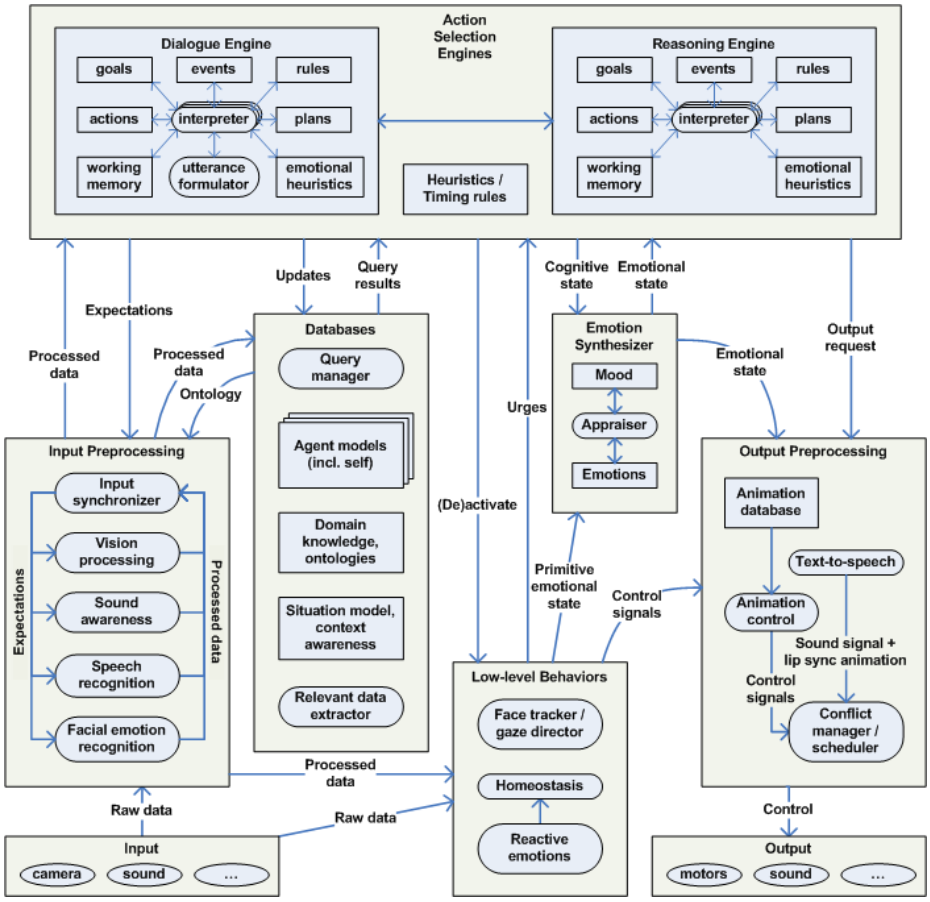

Fig. 1. A generic architecture for a companion robot. The architecture takes into account the possible (or rather, probable) existence of multiple *input modalities*, multiple *input preprocessing* modules for each input modality, *databases* for filtering, storing, and querying relevant information, *action selection engines* for complex, goal-directed, long-term processes such as conversing, planning, and locomotion, an *emotion synthesizer* producing emotions that influence action selection and animations, multiple (reactive) *low-level behaviors* that can compete for output control, multiple *output preprocessing* modules including a conflict manager, and finally, multiple *output modalities*. Straight boxes stand for data storages, rounded boxes for processes, and ovals for sensors/actuators. The interfaces (arrows) between different modules indicate flow of data or control; the connections and contents are made more precise in the text. Note that only the 'ultimate' companion robot would fully implement all depicted modules; a typical companion robot implementation will probably leave out some modules or implement them empty, awaiting future work.

3 Functional Components Constituting the Architecture

In this section we describe the 'blocks' that constitute the proposed architecture. The interfaces ('arrows') between the components are explained in Section 4.

To begin with, the architecture is divided into eight *functional components* (i.e. the larger boxes encompassing the smaller blocks). Each functional component contains several *modules* that are functionally related. Modules drawn as straight boxes represent data storages, the rounded boxes represent processes, and the ovals represent sensors and actuators. Each process is allowed to run in a separate thread, or even on a different, dedicated machine.

No synchronization is forced between these processes by the architecture; they can simply send information to each other (see Section 4), delegating the task of making links between data coming in from different sources to the processes themselves. Below, each of the eight functional components is described, together with the modules they encompass.

Input Modalities A companion robot typically has a rich arsenal of input modalities or sensors. These are grouped in the lower left corner of Figure 1, but only partially filled in. Of course, different kinds of companion robots can have different input modalities, of which a camera and a microphone are probably the most widely occurring. Other sensors may include touch, (infrared) proximity, accelerometer, etc.

Input Preprocessing It is impractical for a reasoning engine to work directly with most raw input data, especially raw visual and auditory data. Therefore, several input preprocessing modules must exist in order to extract salient features from these raw inputs and convert these to a suitable data format. Some input modalities may even require multiple preprocessing modules; for example, one audio processing module may extract only speech from an audio signal and produce text, while another audio processing module may extract other kinds of sounds to create a level of 'sound awareness' for the companion robot. Note that some of these input preprocessing modules may be readily available as off-the-shelf software (most notably, speech recognizers), so a generic architecture must provide a place for them to be plugged in.

Furthermore, there may be need for an input synchronizer that can make links between processed data from different modalities, in order to pass it as a single event to another module. The input synchronizer may initially be implemented empty; that is, it simply passes all processed data unchanged to connected modules. The input synchronizer can also be used to dispatch expectations that are formed by the action selection engines to the input preprocessing modules, which can use these expectations to facilitate feature recognition.

Low-level Behaviors Low-level behaviors are autonomous processes that compete for control of actuators in an emergent way. Some behaviors may also influence each other and other modules. Examples of low-level behaviors include face tracking and gaze directing, blinking, breathing, and other 'idle' animations, homeostasis such as the need for interaction, sleep, and 'hunger' (low battery power), and reactive emotions such as startle and disgust.

Action Selection Engines The 'heart' of the architecture is formed by the action selection engines. These are cognitive-level processes that select actions based on collections of data, goals, plans, events, rules, and heuristics. The outputs that they produce can generally not be directly executed by the actuators,

but will have to be preprocessed first to appropriate control signals. Note that the interpreters of the action selection engines are depicted as layered to indicate that they can be multi-threaded.

The reasoning engine may be based on the BDI theory of beliefs, desires, and intentions [1], deciding which actions to take based on percepts and its internal state. It should be noted that in terms of the BDI theory, the databases component plus the working memories of the action selection engines constitute the robot's beliefs. An action selected by the reasoning engine may be sent to an output preprocessing module, but it can also consist of a request to initiate a dialogue. Because dialogues are generally complex and spread over a longer period of time, a dedicated action selection engine may be needed to successfully have a conversation. This dialogue engine contains an extra process called an utterance formulator; the task of this module is to convert an illocutionary act to fully annotated text, i.e. the exact text to utter together with information about speed, emphasis, tone, etc. This text can then be converted to audio output by the text-to-speech module (in the output preprocessing component).

A similar discussion about separating dialogues and (strategic) planning can be held for locomotion. In our research we have worked with stationary companion robots that focus on dialogues and facial animations. But there can of course be companion robots with advanced limbs and motions. For such robots there may be need for a third action selection engine, dedicated to motion planning. In the proposed architecture, there is room for additional dedicated engines in the functional component of action selection engines.

Finally, the architecture provides for a module called heuristics / timing rules. This is a collection of heuristics for balancing control between the different action selection engines, as they are assumed to be autonomous processes. The different engines will get priorities in different cases; e.g., the plans of the dialogue engine will get top priority if a misunderstanding needs to be repaired. On the other hand, if the dialogue engine does not have any urgent issues, the reasoning engine will get control over the interaction in order to address its goals. Furthermore, it can verify whether the goals of the different action selection engines adhere to certain norms that apply to the companion robot in question, as well as provide new goals based on timing rules; e.g., to avoid long silences, the robot should always say something within a few seconds, even if the reasoning engine is still busy.

Databases We have created a distinct functional component in the architecture where data is stored in different forms. This data includes domain knowledge, ontologies, situation models, and profiles of the robot itself and of other agents. The ontologies and domain knowledge are (possibly static) databases that are used by the input preprocessing modules to find data representations suitable to the action selection engines and databases. The agent profiles store information about other agents, such as the robot's interaction histories with these modeled agents, the common grounds between the robot and each modeled agent, and the presumed beliefs, goals, plans, and emotions of each modeled agent. These agent models also include one of the robot itself, which enables it to reason about its own emotions, goals, etc.

In order to provide a consistent interface to these different databases, a query manager must be in place to handle queries, originating from the action selection engines. A special situation arises when the robot queries its own agent model, for there already exist modules containing the goals, plans, and emotions of the robot itself. So the query manager should ensure that queries concerning these types of data can get their results directly from these modules.

Finally, a relevant data extractor takes care of interpreting incoming data in order to determine whether it can be stored in a more suitable format; e.g., if visual and auditory data from the input preprocessing component provides new (updated) information about the environment of the robot, it is interpreted by the relevant data extractor and stored in the situation model. Moreover, simple spatial-temporal reasoning may be performed by the relevant data extractor. If advanced spatial-temporal reasoning is needed for some companion robot, it may be better to delegate this task to a separate input preprocessing module.

Emotion Synthesizer Typically, companion robots must show some level of affective behavior. This means responding appropriately to emotions of a (human) user, but also includes experiencing emotions itself in response to the current situation and its internal state. The emotions that concern this functional component are those of the companion robot itself and are at a cognitive level, i.e., at the level of the action selection engines. Examples of emotions are joy when a goal is achieved, disappointment when a plan fails, resentment when another agent (e.g. a human user) gains something at the expense of the robot, etc. More reactive emotions (e.g., startle) can be handled by a low-level behavior.

The emotion component consists of three parts. The appraiser is a process that triggers the creation of emotions based on the state of the action selection engines. The intensity of triggered emotions is influenced by the robot's mood (the representation of which may be as simple as a single number) and a database of previously triggered emotions. This database of emotions then influences the action selection engines (by way of their emotional heuristics module) and the animations of the robot, e.g., by showing a happy or sad face.

Output Preprocessing Different processes may try to control the robot's actuators at the same time; obviously, this calls for conflict management and scheduling of control signals. Moreover, some modules may produce actions that cannot be directly executed, but instead these abstract actions need some preprocessing to convert them to the low-level control signals expected by the robot's actuators. E.g., the dialogue engine may want some sentence to be uttered by the robot, but this must first be converted from text to a sound signal before it can be sent to the loudspeaker. This functionality is provided by the text-to-speech module, which is also assumed to produce corresponding lip sync animations.

For companion robots with a relatively simple motor system, it suffices to have a single module for animation control which converts abstract animation commands to low-level control signals. This can be done with the help of an animation database containing sequences of animations that can be invoked by name and then readily played out. For companion robots with a complex motor system, the animation control module may be replaced by a motion engine (which

is placed among the other action selection engines), as discussed above. In this case, an animation database may still fulfill an important role as a storage of small, commonly used sequences of motor commands.

Finally, actuator control requests may occur concurrently and be in conflict with each other. It is the task of the conflict manager to provide the actuators with consistent control signals. This can be done by choosing between conflicting requests, scheduling concurrent requests, or merging them. These choices are made on a domain-dependent basis.

Output Modalities All output modalities or actuators are grouped in the lower right corner of Figure 1. Similarly with the input modalities, these will be different for different kinds of companion robots, but a typical companion robot will probably have at least some motors (for e.g. facial expressions and locomotion) and a loudspeaker. Other actuators may include lights, radio, control of other electronic devices, etc.

4 Interfaces Between Functional Components

In this section, we explain the meaning of the interfaces between the functional components. For cosmetic reasons, the 'arrows' in Figure 1 appear to lead from one functional component to another, while they actually connect one or more specific modules *inside* a functional component to other modules inside another functional component. References to arrows in Figure 1 are marked in boldface.

Raw data that is obtained by the input sensors is sent to the input preprocessing component for processing. Needless to say, data from each sensor is sent to the appropriate processing module; e.g., input from the camera is sent to the vision processing and facial emotion recognition modules, while input from the microphone is sent to the sound awareness and speech recognition modules. Any module inside the low-level behaviors component is also allowed to access all raw input data if it wants to perform its own feature extraction. In addition to raw data, low-level behaviors also have access to the **Processed data** from the modules inside the input preprocessing component. After the processed data is synchronized (or not) by the input synchronizer, it is sent to the action selection engines, where it is placed in the *events* modules inside the engines. The processed data is also sent to the databases, where the relevant data extractor will process and dispatch relevant data to each of the databases; e.g., context-relevant features are added to the situation model, while emotions, intentions and attention of a user that are recognized by the various input preprocessing modules are put in the appropriate agent model. Furthermore, the action selection engines can form **Expectations** about future events. These expectations are sent from the action selection engines back to the input synchronizer, which subsequently splits up the expectations and sends them to the appropriate input processing modules. They can then use these expectations to facilitate processing of input.

All processing modules in the input preprocessing component have access to **Ontology** information, which they might need to process raw data properly; e.g., the vision processing module might need ontological information about a perceived object in order to classify it as a particular item. This also ensures

the use of consistent data formats. The processing modules can obtain this ontological information via the query manager in the databases component, which takes care of all queries to the databases. **Updates** to the databases can be performed by the action selection engines. The updates are processed by the relevant data extractor, which places the data in a suitable format in the appropriate database, in the same way as the processed data from the input preprocessing component. **Query results** can be requested by the action selection engines from the databases. The query manager processes the query and searches the proper database(s), guaranteeing a coherent interface to all databases.

(De)activate signals can be sent from the action selection engines to the low-level behaviors component. These signals allow the action selection engines some cognitive control over the robot's reactive behavior; e.g., if needed, the face tracker can be activated or deactivated, or in some special cases the reactive emotions can be turned off. **Urges** arising from the low-level behaviors can be made into goals for the action selection engines. For example, if the homeostasis module detects a low energy level, a goal to go to the nearest electricity socket can be added to the goals of the motion engine.

The action selection engines provide their **Cognitive state** to the emotion synthesizer. The cognitive state can be used by the appraiser to synthesize appropriate emotions. In addition to the cognitive state, a **Primitive emotional state** is also sent to the appraiser, where it can influence the intensity of cognitive-level emotions and the robot's mood. The current **Emotional state**, which is a compilation of the collection of triggered emotions, is sent to the action selection engines. The emotional heuristics inside the action selection engines then determine how the interpreter is influenced by these emotions. The animation control module inside the output preprocessing component also receives the emotional state of the agent, so that it can select a suitable facial expression from the animation database representing the current emotional state.

Output requests are sent from the interpreters inside action selection engines to the output preprocessing component. Of course, the different kinds of output requests are sent to different modules inside the output preprocessing; e.g., (annotated) utterances from the dialogue engine's utterance formulator are sent to the text-to-speech module, while any actions from the action selection engines that involve motors are sent to the animation control module. The synchronization of all output signals is taken care of by the conflict manager, as explained in the previous section. Finally, **Control** signals are gathered and synchronized by the conflict manager inside the output preprocessing component and sent to the appropriate output modality.

5 Discussion and Related Work

We do not claim that the presented architecture is perfect, and although we claim that it is generic for companion robots, it is probably not unique. Another team setting out to make a generic companion robot architecture will probably come up with a different figure. However, we expect the level of complexity of alternative architectures to resemble that of the one presented here, as it takes

many components and processes to achieve reasonably social behavior. It should be noted that the intelligence of the system may not lie within the modules, but rather in the wiring (the "arrows"). The presentation of an architecture should therefore include a discussion on the particular choice of interfaces between modules. We draw confidence in our architecture from the fact that mappings can be found between this one and the architectures of existing companion robots and affective virtual characters, several of which we discuss next.

Breazeal [2] uses competing behaviors for the robot Kismet in order to achieve an emerging overall behavior that is sociable as a small child. Kismet has a number of different response types with activation levels that change according to Kismet's interaction with a user. In Kismet's architecture, the Behavior System and Motivation System can be mapped on our low-level behaviors; however, it lacks cognitive reasoning (obviously this was not necessary for its application), which is provided by our action selection engines. Other components in Kismet's architecture pertain to input and output processing, which map to corresponding preprocessing modules in our architecture.

Max, the "Multimodal Assembly eXpert" developed at the University of Bielefeld [4], can also be mapped to our architecture. For example, it uses a reasoning engine that provides feedback to the input module to focus attention to certain input signals, which is similar to our *expectations*. It also has a lower-level Reactive Behavior layer that produces direct output without having to enter the reasoning process, and a Mediator that performs the same task as our conflict manager (i.e. synchronizing output). However, Max only has one (BDI) reasoning engine, where we have provided for two or more action selection engines.

6 Conclusions and Future Research

In this paper, we have presented a generic architecture for a companion robot. We do not claim that it should be the foundation of the 'ultimate' companion robot; rather, we have presented this architecture in order to make many of the issues encountered when programming a companion robot explicit, so that these issues can be appropriately discussed.

The implementation of our companion robot architecture is yet to be finished, although it should be noted that it does not have to be fully implemented in all cases. Some of the functional components can be left out, simplified, or even extended (depending on the application) or programmed empty (awaiting future work), while for some modules, off-the-shelf or built-in software can be used. We are using the Philips iCat [3] as a platform for developing a proof of concept of the proposed architecture. Ultimately, it can be used as a framework on top of which the software of new companion robots can be developed.

References

1. Bratman, M., *Intention, Plans, and Practical Reasoning*, Harvard U. Press, 1987.
2. Breazeal, C.L., *Designing Sociable Robots*, MIT Press, 2002.
3. Van Breemen, A.J.N. *iCat: Experimenting with Animabotics*, AISB 2005 Creative Robotics Symposium, England, 2005.
4. Kopp, S., Jung, B., Lessmann, N., Wachsmuth, I. *Max – A Multimodal Assistant in Virtual Reality Construction.* In KI-Künstliche Intelligenz 4/03, p. 11-17, 2003.