Methods for Complex Single-Mind Architecture Designs (Short Paper)

Kristinn R. Thórisson
CADIA / Dept. Comp. Science
Kringlunni 1
103 Reykjavik, Iceland
+354 898 0398
thorisson@ru.is

Gudny Ragna Jonsdottir
CADIA / Dept. Comp. Science
Kringlunni 1
103 Reykjavik
Iceland
gudny04@ru.is

Eric Nivel
Center For Analysis & Design of Intelligent Agents
Kringlunni 1
103 Reykjavik, Iceland
eric@ru.is

ABSTRACT
The implementation of software systems with large numbers of heterogeneous components calls for a powerful design methodology. Although several such methodologies have been proposed, many lack application to construction of single-mind systems. We have employed the Constructionist Design Methodology (CDM) in building several such systems, including an autonomous radio show host. Proposing modules communicating through messages via blackboards as key building blocks for interactive intelligences, the methodology has been of considerable help in the early stages of designing several large architectures. This paper describes efforts to extend the CDM with more detailed support for the modularization process. We detail our use of a combination of abstraction and finite state machines in modularizing the realtime turntaking system of the radio show host. Our experience shows considerable benefits and added flexibility in the creation of large architectures when using the new modularization principles.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Modules and interfaces.* I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Coherence and coordination, Intelligent agents.* I.6.5 [**Simulation and Modeling**]: Model Development – *Modeling methodologies.*

General Terms
Algorithms, Performance, Design, Experimentation, Theory.

Keywords
Methodology, design, architecture, speech, interactive, realtime, communicative humanoids, single-mind architectures.

1. INTRODUCTION
The creation of realtime dialogue systems containing a large set of integrated functionalities requires a strong development methodology. Unfortunately, many of the methodologies that have been offered for this purpose, e.g. multi-component and agent-based simulation and modeling (cf. [6,7]), suffer from lack of actual experience in artificial intelligence projects, especially where the focus is the construction of single-mind systems, e.g. robotics or autonomous radio show hosts – that is, systems where a single external body has to be managed as a limited resource. Such systems put strong – and unique – requirements on coordination and temporal adequacy in the architecture.

Modularity is a highly desirable feature of any complex system as it keeps transparency and openness in the architecture. However, it means decoupling of components, which means increases in coordination between them. Thus, the *kind* of modularity and methodology one adopts is key. One our goals is to explore how true flexibility can be obtained in the development of large evolving systems. General software engineering methods are appropriate when the target system is clearly and correctly specified, as is the case is standard IT development. This approach can turn into a burden, however, when the target system is built incrementally and experimentally, as continual expansion often changes the system's very specification. The majority of methodologies intended for agent-based systems, e.g. MaSE [6], which one might expect to do better in this respect, do not address the need for strong iterative design either. The Constructionist Design Methodology (CDM, [4]), however, was specifically created to help build interactive A.I. systems addressing, in particular, the problems associated with iterative design for large heterogeneous agent-based systems with complex information flow. We are in the process of building a fully autonomous radio show host. One of its tasks will be to interview people over the phone. The current system is a (growing) collection of over 40 (semi-autonomous) modules that communicate (1-to-n) via discrete messages; each module having non-trivial operational logic and interactions. In this paper we propose an extension to the modularization principles of the CDM and evaluate them in the context of the development of a subcomponent of the radio show host system, namely, its turntaking system.

Following the CDM, an initial rough design sketch of the turntaking system was fleshed out over three iteration steps (the turntaking system itself will be described in a future paper); the new modularization principles were applied and evaluated during development the process. The results show that the new principles are highly useful in keeping complexity to manageable proportions and allowing continual expansion of the system, including dynamic changes in design requirements.

Cite as: Methods for Complex Single-Mind Architecture Designs (Short Paper), K. R. Thórisson, G. R. Jonsdottir & E. Nivel, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.

2. CDM

The Constructionist Design Methodology has been applied in the construction of several interactive systems (cf. [2,4]). CDM proposes using publish-subscribe techniques and blackboards as the backbone for integration, as this has significant benefits (see e.g. [4]) in systems that need to be experimentally and incrementally built. CDM proposes 9 principles (semi-independent steps) to help with such development; the full set is described in [4]. Here we will zoom in on the principles relating to iterative modularization, (CDM principles 3a and 3c) in relation to iteration (CDM principle 5). The modularization principle in the CDM states:

> ***CDM principle 3.a****: Classify modules into one of three roles, according to whether they serve perception, decision/planning, or action/animation. Each of these roles can be further split into more levels. For example, high-level perception versus low-level perception, short-, mid- and long-term planning, and high-level animation versus low-level animation.*

We have found the use of abstraction levels to be indispensable, but how abstraction is used in architectural design is neither obvious nor well understood. The guidelines in the CDM for using levels is short; here we put special focus on the construction of modules and messages in relation to abstraction levels.

CDM's *iterate* principle (#5) states that several of the principles, including 3a above, must be iteratively applied in the process of building a full, complete system. It plays an especially important role in the modularization process, as modules define messages and messages reciprocally define modules. Here we show how this principle interacts with 3a as we expand the architecture.

3. THE RADIO SHOW HOST SYSTEM

Elsewhere we have presented the view that turntaking and natural dialogue is a complex system that is best modeled as (a large number of) interacting modules [10]. Following CDM principle *3a*, we started at the low level with several perception modules for extracting prosodical information from a person's speech. Then we expanded the system with a set of control/decision modules that, based on the perceptual processing output of the perception modules, decided which turntaking state is most likely ("I have turn", "other has turn", etc., see [3] for details). The current system is an intermediate state between the basic starting point (based on [3]) and what we intend to build for a fully-fledged radio show host. At present the system contains the following functionalities, each of which is represented by one or more modules: **Pitch Tracker** (low-level perception): From the raw audio signal, identifies pitch, pitch derivative, speech on/off, silences and hums. **Pitch Analyzer** (mid-level perception): Analyzes pitch in more detail, as well as overlapping speech and silences. **Speech Recognizer** (high-level perception): Translates audio signal to text (continuous speech recognition). **Interpreters** (high level perception): Several perception modules get input from a single Speech Recognizer; each interpreter has its specified task, e.g. look for nouns, dates etc. **Interpretation Director** (high-level perception): Receives input from Content Planner (see below) on what to look for and analyses the output of all Interpreters based on that information. **Content Planner** (high-level decision): Composes *what* to say, based on inner goals and information from the ID. It includes knowledge bases for creating semantic responses. **Dialogue Planner** (high-level decision): Delivering the next "executable thought unit", as provided by the Content Planner, producing fillers or gracefully giving turn when content is not available from the CP within acceptable time limits; takes input from perception and internal states, e.g. *motivation to speak*; keeps a conversation going (as is possible without information about the topic). **Action Generation** (Speech Synthesizer, mid-level action): Executes content provided by CP when the Turntaking and DP systems claim that the system has the turn.

Turntaking System (low-level decision): This is the key system that we will discuss here. It is composed of roughly 20 modules that are responsible for the realtime delivery of speech acts by the radio show host. It assesses the dialogue state based on input from perceptual modules. In this system various perception and decision mechanisms related to turntaking are grouped via global states that describe which perceptions and decisions are relevant at any point in time, and thus they represent the disposition of the system at any point in the dialogue, e.g. whether it is appropriate to expect a certain turntaking cue to be produced, whether it is relevant to generate a particular behavior (e.g. volume increase in the voice upon interruption by the other), etc.

4. DESIGN ITERATIONS

One of our goals for the radio show host is to enable it to do interesting interviews with listeners, which calls for a powerful, dynamic turntaking (TT) system enabling it to deal with hesitations, interruptions and various other features of fast-paced on-air interviewing. The use of FSMs is prevalent in current engineering practice; more often than not, FSMs limit the ability to expand a system as needed as additions and extensions require significant reengineering of what was built before; methodologies relying on FSMs generally work against dynamic expandability.

Version 1. Following CDM, we started with a basic version of the turntaking system (based on [3]), using a finite state machine (FSM) to represent the main states of the dialogue, e.g. "I have turn", "Other gives turn", etc. These were implemented as semi-global state variables with standard transition rule design; transitions are controlled by perceptual modules monitoring the pitch, silences and filler words ("uhm", "ahh") of the interviewee (in this phase the only perception used was pause duration). The Content Planner (CP) system generated utterances to be spoken and when the TT system deemed that the agent had the turn the Dialogue Planner (DP) commanded the speech synthesizer to say the next sentence(s). This architecture is depicted in *Fig. 1*.

The result of this phase was thus a "polite" but dumb and reactive TT system that can interact with a human relatively reliably, perceiving appropriate turn transition points (based on the analysis done by perceptual modules), and decide to either give or accept turn (depending on whether it had something to say). Reliable performance of this version was reached with pause threshold set to 1-1.2 seconds, that is, the system waits for a pause of that duration before assuming that it can take the turn. This is quite a bit slower than the average turntaking speed of a human speaker (200-300 msec [1]).

Version 2. Next we wanted to give the radio show host the additional ability to both want turn (e.g. if it has something important to say) and perceive that other wants turn when it is speaking. Among other implications these modifications meant that new states need to be added to the existing FSM. Generally, FSMs are difficult to expand because each change may affect a

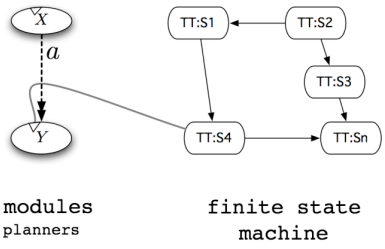

Figure 1. Partial view of the initial turntaking system. An FSM represents valid dialog states; module *X* produces next thing to say and sends it (*a*) to a speech synthesizer module *Y*; *Y* is only active when the radio show host has the turn (*TT:S4*).

significant part of the system and version 2 modifications called for new states, new transitions, as well as modified older transitions. Yet we wanted to keep the current system. A better method for expanding the (otherwise useful) FSM was therefore sorely needed.

We employed a different way of handling the state transitions in the FSM: Instead of monolithic, integrated *state+transition* rules, we decoupled the states and the transitions via the *modules+messages* paradigm. In this scheme modules handle the transitions by posting state transition messages to a global state system (via middleware). With this solution adding a new transition, whether it had been anticipated by the designers or not, requires simply adding one (or more) module(s) that trigger that particular transition (*Fig. 2*).

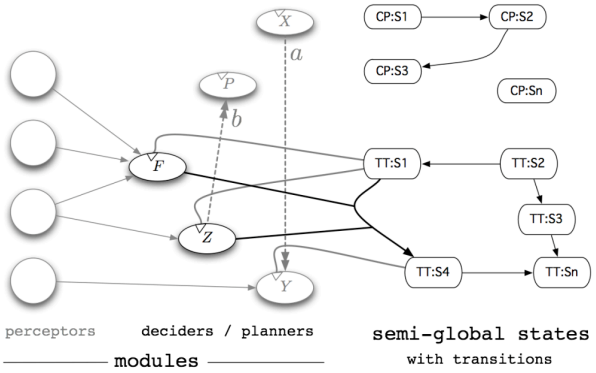

Figure 2. States and their transition rules are decoupled via transition modules that change states via messages. Here, *F* and *Z* (only active during *TT:S1)* control the transition from *TT:S1* (*other-gives-turn*) to *TT:S4* (*I-have-turn*) independently of each other.

The result was a more "tolerant" radio host – it can now also yield the turn if deemed appropriate and tries to indicate politely whenever it wants the turn (e.g. by producing fillers). The new design methodology allowed us to significantly improve the system's speed of taking turns, with minimal modifications: (i) the addition of one new state, (ii) the addition of a single module for controlling the transition to it and (iii) configuring one of the old modules to be active in the new state. This allowed a reduction from 1 second to 400 msec for pause threshold, tripling the system's ability to take turn quickly. This version, however, is still missing significant functionality, such as being able to interrupt. Moreover, the "flat" architecture with a "pool" of modules suggests no constraints on which modules are allowed to connect to where; we know from experience that "free-form" additions of connections will very quickly turn the system into unmanageable spaghetti. More detailed design principles are needed.

Version 3. Next we wanted the host to be able to interrupt the user. A motivational factor, *urge to speak,* would determine whether the host should take turn and start speaking (no matter what its perceptual, decision and TT states were saying). This addition meant that either the TT FSM would be modified with a state transition from every current state to the I-have-turn state, or the TT system could be allowed to be interrupted from the outside by another system, external to its FSM-based mechanism. We chose the latter option: The Dialogue Planner (DP) was given the extra role of modifying the execution of the TT FSM. We assigned the DP to a higher abstraction layer than the turntaking system (*Fig. 3*). A new module (*A*) was added to control TT-relevant state switching messages in the TT FSM, introducing an externally-activated "state jump" (*c*). The DP thus simply added transition paths to the existing system, other things staying exactly the same. With the DP system at a more abstract level we managed to expand the whole system, keeping connections between the TT and DP to a minimum.

The resulting radio show host turntaking system still has all the functionality of its previous versions, being able to take turns relatively quickly, but now with the added capability of being able to "choose to be impolite" based on a higher-level motivation, taking turn even when the other person doesn't want to give it. Meanwhile the TT system *retains its original modularity*, and it is open enough to allow external steering.

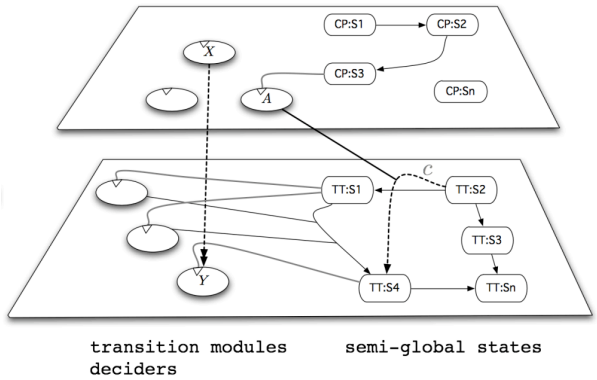

Figure 3. Functions related to longer time scales and higher-level concepts (upper plane) are separated from more reactive parts of the system (lower plane).

A feature of the development of hierarchical layers of modules, according to the CDM, proposes using decomposition to provide a certain degree of plasticity in the module arrangement: The degree of plasticity we obtained in this construction process – and thus in the resulting overall system – is directly related to the degree of openness of modularization at each given stage of development. This is a clear advantage over other approaches. The advantages became even clearer when we wanted the system to automatically adapt to its interlocutor over time. Implementing distributed learning algorithms within each of the existing modules would be a rather complex task. Following the CDM we created a separate *Learner* module; this is the only module that needs to know a learning method (currently Q-Learning), providing abstract data (patterns of action/consequence relations) as output. It does not

make any assumption on subsequent usage of this information. Symmetrically, modules consuming this data do not need to know how it has come about (why action *x* is better than action *y* in state *z*), they only have to perform *e-greedy* search on the provided state space for the best action to take at any point in time.

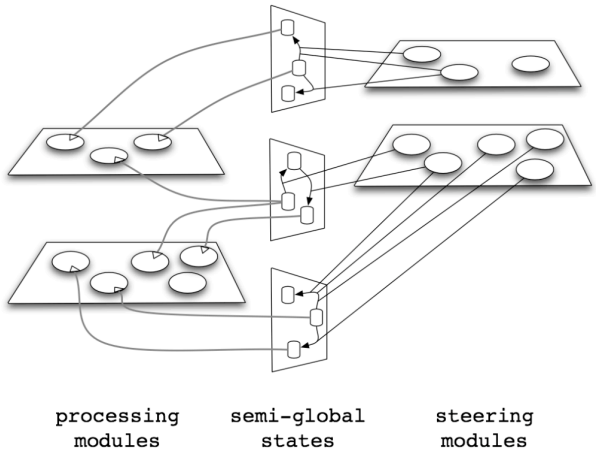

Figure 4: Modules are arranged in abstraction layers, and are given structural roles, *processing* and/or *steering*. These modules modify shared system states while their interaction span across the layers.

5. MODULARIZATION OF STATES

In our approach system expansion was first achieved by using an explicit, (semi)global FSM. We then applied modularization principles to the modules' interaction space itself – i.e. the FSM – and distributed states in another dimension, orthogonal to the dimension of module layering. A subset of the system states represents the "skeleton" of a given behavior, defining a natural partition in the entire set of related states (*Fig. 4*). This partitioning is *orthogonal to the abstraction hierarchy*, and depicts the natural *indirect coupling* of modules via transitions along the state graph.

To support FSM-based designs in the CDM we introduce the following principle: ***Orthogonal partitioning of system states: Explicitly-represented states are grouped and partitioned in separate blackboards, each serving the role of an interaction space for modules in charge of steering a particular behavior of the system.*** Note that a "behavior" applies to both overt and covert actions, allowing complex couplings between internal and external control and event mechanisms.

In the scheme we are proposing, using a combination of FSMs and message passing, modules are automatically activated upon the transition to a state they are registered for, and state switches are executed immediately upon reception of a state transition message. This has potential for race conditions: Be the system in state A, transitioning to B on message T_{AB} and C on T_{AC}; if module M_1 fires T_{AB} and – concurrently – M_2 fires T_{AC}, then the first module given the CPU, say M_2, will determine the next state, C, invalidating the transition message posted by the unlucky module M_1. In our approach, if concurrent steering of local system states is needed, we use dedicated modules responsible for module activation; such modules resolve conflicting concurrent state transitions and send *final* state transition messages. In this way modules realizing transitions do not send state transition commands but transition requests to the conflict resolvers. This way the modularization of the interaction space proposed above remains valid and without race condition problems.

6. CONCLUSION

To date the CDM has been useful for guiding our designs. The results based on the work described here show that the new modularization methods help extend the present CDM principles, allowing even larger architectures to be built iteratively; we have grown the current system to more than 60 highly complex, heterogeneous modules, exchanging over 100 message types in a dynamic 1-to-many arrangement. These new methods help ensure that continual design refinements can continue at a low cost in complex systems. Displacing the complexity from "fat black-boxes" to more numerous – but smaller – modules collaborating to steer the overall system state, according to our principles, gives the designer the extra level of plasticity needed for build-run-experiment-augment iteration cycles.

7. REFERENCES

[1] Ford, C. and S. A. Thompson (1996). Interactional units in conversation: Syntactic, intonational, and pragmatic resources for the management of turns. In E. Ochs, E. Schegloff and S. A. Thompson (Eds.), *Interaction and Grammar*, 134-184, Cambridge University Press.

[2] Ng-Thow-Hing, V., T. List, K.R. Thórisson, J. Lim, J. Wormer (2007). Design and Evaluation of Communication Middleware in a Distributed Humanoid Robot Architecture. *IROS '07 Workshop Measures and Procedures for the Evaluation of Robot Architectures and Middleware,* 29 Oct. - 2 Nov. San Diego, Califronia, U.S.A.

[3] Thórisson, K. R. (2002). Natural Turn-Taking Needs No Manual: A Computational Model, From Perception to Action. In B. Granström, D. House, I. Karlsson (eds.), Multimodality in Language and Speech Systems, 173-207. Dodrecht, The Netherlands: Kluwer Academic Publishers.

[4] Thórisson, K. R., H. Benko, A. Arnold, D. Abramov, S. Maskey, A. Vaseekaran (2004). Constructionist Design Methodology for Interactive Intelligences. *A.I. Magazine*, **25**(4): 77-90. Menlo Park, CA: American Association for Artificial Intelligence.

[5] Thórisson, K. R. (2008). Modeling Multimodal Communication as a Complex System. In I. Wachsmuth & G. Knoblich (eds.), *Modeling Communication with Robots and Virtual Humans*, LNAI **4930**, 143-168. New York: Springer.

[6] Wood, M. F. and S. A. DeLoach (2001). An Overview of the Multiagent Systems Engineering Methodology. In *Agent-Oriented Software Engineering*, LNCS, vol. **1957**. Berlin: Springer, January, 207-221.

[7] Wooldridge, M., Jennings, N. and Kinny, D. (2000). The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, **3**(3):285-3.