Automating spoken dialogue managament design using machine learning: An industry perspective

Tim Paek & Roberto Pieraccini

Liisa Vaht

Dialogue management

Automated systems using speech technology rely on DM

Fundamental task: decide what to do with user input

-> define the next action

Actions directly impact the user

-> largely responsible for the system performance and user experience

Automating the DM strategy design could enable faster development of advanced commercial applications

Commercial systems

Application domains:

-> call centre automation,

-> automated troubleshooting,

-> entertainment systems

-> ...and many more

Focus is laid on:

-> industry performance metrics (ROI, automation rate)

-> performance, usability

Commercial development

Follows a certain development process and rules

-> the development cycle starts with requirements collection

-> functional specification

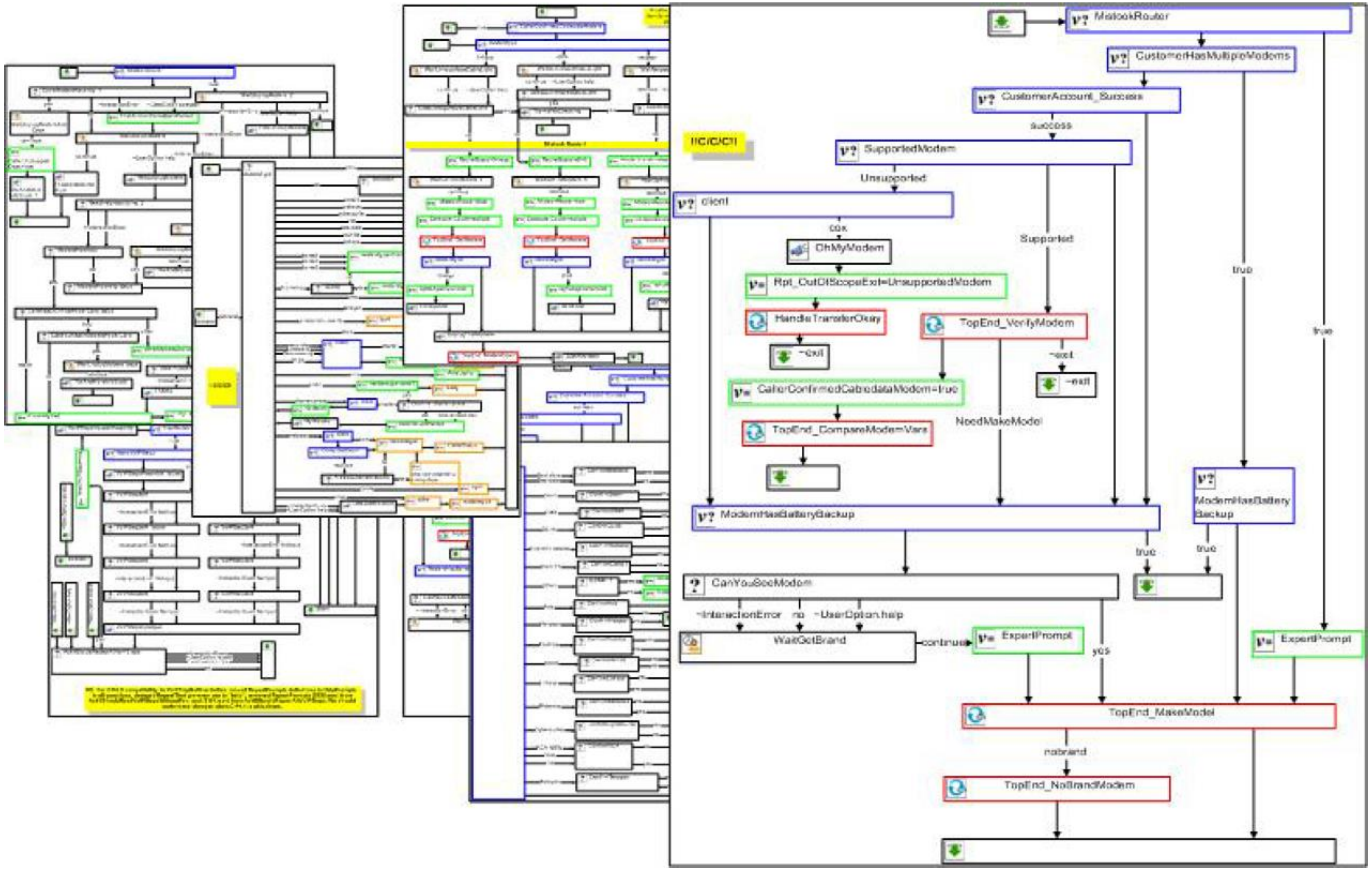
-> VUI design (typically determined by a call-flow graph)

-> development and tuning of grammars and language models

-> backend integration (interaction with systems such as CRM)

-> error handling (establishing a confidence level)

-> testing and monitoring, which can last the whole lifetime of the application


MstookRouter
CustomerHasMultipleModems
CustomerAccount_Success
success
!!C/C/C!!
SupportedModem
Unsupported
client
cox
OhMyModem
Rpt_OutOfScopeExit=UnsupportedModem
HandleTransferOkay
~exit
Supported
TopEnd_VerifyModem
~exit
~exit
CallerConfirmedCabrodataModem=true
TopEnd_CompareModemVars
NeedMakeModel
true
true
ModemHasBatteryBackup
ModemHasBattery Backup
true
true
CanYouSeeModem
~InteractionError
no
~UserOption.help
WaitGetBrand
continue
ExpertPrompt
yes
ExpertPrompt
TopEnd_MakeModel
nobrand
TopEnd_NoBrandModem

VUI completeness

All possible combinations of user inputs and conditions need to be anticipated

“no unpredictable user input should ever lead to unforeseeable behaviour”

-> only two outcomes are acceptable: user task is completed or a fall-back strategy activated

Requirement for all commercial applications

-> developers need to guarantee VUI completeness to their customers

-> they system has to perform exactly as defined in VUI specification document

and will be tested according to it

Current state

Developers tend to rely on accomplished tools and best practices

Call-flow graphs limit the types of dialogue interaction possible

-> but are easy to understand and adjust

-> directly match the requirements of the VUI specification

Developers need to hand craft system responses to all possible user input

-> extremely challenging, labour intensive and error-prone process

Researchers have been turning to machine learning methods to optimize the process

Reinforcement learning

Automating design of DM strategies

Specifies *what* to achieve, not *how* to achieve it

-> maximize its total reward received

System can learn from the rewards it receives

->learns from user feedback data

-> optimal policy is derived

Interaction is represented as a fully or partially observable Markov decision process

-> *(S, A, T, R)*

Reward function *R*

Typical practice: assign a small negative reward for each dialogue turn and a large negative or positive reward upon completing the interaction

Overall behaviour of the system is very sensitive to changes in *R*

Not easily adjusted

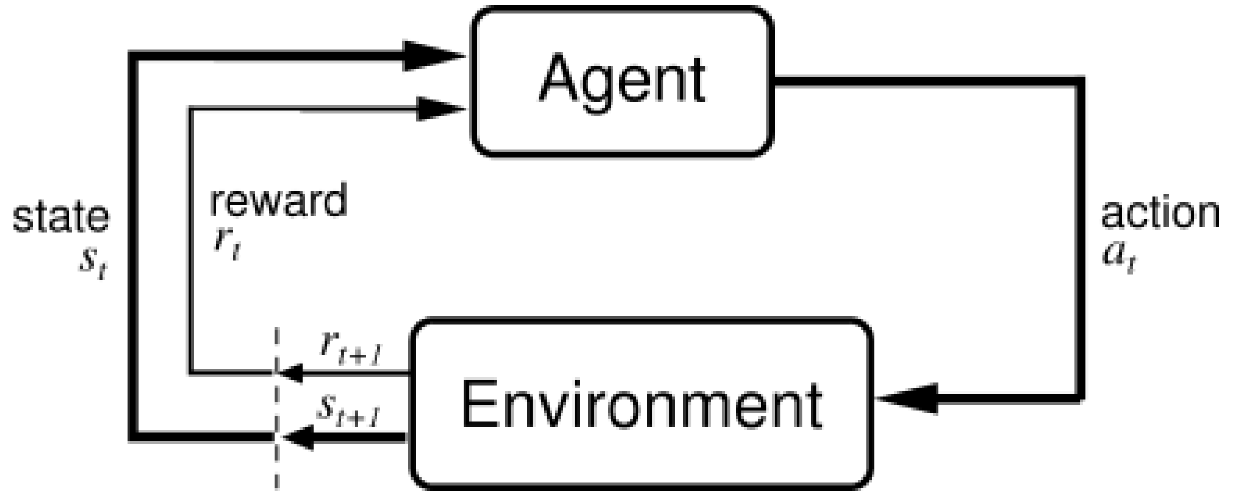
Assigning the value of *R* requires very good understanding of it

-> mostly hand-crafted and tuned

-> set to be static

->might be beneficial to adapt according to user type and or goal

Reinforcement learning agent



Agent receives the state and reward signals from the environment

To decide what to do, the agent compares different sequences of rewards

Environment receives the action signal from the agent

Policy

"The ultimate product of utilizing reinforcement learning methods to DM"

-> learned from the data

Other components serve only to change and improve the policy

Decision making function, prescribes an optimal action the SDS should take

-> limited on the data it is trained

-> exploration vs exploitation dilemma without having explored all actions in all states

"A black box that tells the system what to do"

-> leaves developers little control

-> not immediately clear how the system will behave

-> complicate to adjust to achieve desired behaviour

Evaluation and training

User simulation

-> learn a generative model of the user to simulate responses

-> time-consuming and impractical to use real users

-> provides a testing environment for conducting controlled experiment

-> generates data for learning policies

But:

-> coming up with realistic training data takes a lot of work

-> user simulations are not fully domain-independent

Drawbacks (from industry perspective)

More difficult to incorporate other goals that developers have

-> such as maintainability, usability of the interface

Developers need to understand how the relative values of R influence the overall behaviour of the system

Fixing some particular action requires adjusting complicated parameters

-> would be straightforward if represented for example in a call-flow graph

Introducing new state variables after deployment is complex and requires the policy to be re-learned

Control vs. automation trade-off

Automation can relieve developers from difficult challenges and improve performance

To guarantee VUI completeness, developers need to know how to control and modify the system

"Give application developers enough control to allow them to ensure VUI completeness"

One way to achieve it is to offer macro level control and fine-grain automation

-> developers are relieved of some difficult challenges

-> can still achieve VUI completeness

Potential areas of automation

Task independent behaviours

-> such as error correction and confirmation behaviour

-> e.g. escalation strategy

Task-specific behaviours

-> such as logic associated with certain customer-care practices

-> e.g. troubleshooting or other that require knowledge and experience

Task-interface behaviours

-> such as prompt selection

-> e.g. selecting the best performing of competing prompts

Conclusion

Developers must guarantee VUI completeness to their customers

Developing a commercial application that achieves this is a demanding process

RL has a lot of potential to take workload off developers and to improve performance,
-> has not truly found its way to commercial applications

In order for RL to be of practical benefit to commercial developers, it has to overcome some challenges

Currently, a good approach would be to use RL for smaller tasks and give the developers macro-level control over the system

What is your take on the control vs. automation trade-off?

Thank you!