

CLASSiC

D1.1.3: Adaptive automata-based DM component, final prototype

Romain Laroche, Ghislain Putois, Philippe Bretier

Distribution: Consortium

CLASSiC

Computational Learning in Adaptive Systems for Spoken Conversation
216594 Deliverable 1.1.3

February 2011



Project funded by the European Community
under the Seventh Framework Programme for
Research and Technological Development



The deliverable identification sheet is to be found on the reverse of this page.

Project ref. no.	216594
Project acronym	CLASSiC
Project full title	Computational Learning in Adaptive Systems for Spoken Conversation
Instrument	STREP
Thematic Priority	Cognitive Systems, Interaction, and Robotics
Start date / duration	01 March 2008 / 36 Months

Security	Consortium
Contractual date of delivery	M36 = February 2011
Actual date of delivery	February 2011
Deliverable number	1.1.3
Deliverable title	D1.1.3: Adaptive automata-based DM component, final prototype
Type	Prototype
Status & version	final 1.0
Number of pages	8 (excluding front matter)
Contributing WP	5
WP/Task responsible	WP1, leader Cambridge University
Other contributors	
Author(s)	Romain Laroche, Ghislain Putois, Philippe Bretier
EC Project Officer	Philippe Gelin
Keywords	Platform, System, user feedbacks, logging, design alternatives, Dialogue Management

The partners in CLASSiC are:

Heriot-Watt University	HWU
University of Cambridge	UCAM
University of Geneva	GENE
Ecole Supérieure d'Electricité	SUPELEC
France Telecom/ Orange Labs	FT
University of Edinburgh HCRC	EDIN

For copies of reports, updates on project activities and other CLASSiC-related information, contact:

The CLASSiC Project Co-ordinator:
Dr. Oliver Lemon
School of Mathematical and Computer Sciences (MACS)
Heriot-Watt University
Edinburgh
EH14 4AS
United Kingdom
O.Lemon@hw.ac.uk
Phone +44 (131) 451 3782 - Fax +44 (0)131 451 3327

Copies of reports and other material can also be accessed via the project's administration homepage,
<http://www.classic-project.org>

No part of this document may be reproduced or transmitted in any form, or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission from the copyright owner.

Contents

Executive Summary	1
1 Introduction	2
2 Reinforcement Learning	2
2.1 Basic Ideas	2
2.2 The design architecture	2
2.3 Application Programming Interface (API)	4
3 Uncertainty Management	4
4 Remarks	6
5 Software	7

Executive Summary

This document is a short report to accompany the Prototype Deliverable D1.1.3, due at month 36 of the CLASSIC project. This prototype is an enhancement of the industrial Dialogue Manager with another module called the Learning Manager that provides the Spoken Dialogue System with adaptive online reinforcement learning capabilities. This document reviews the foundations of the reinforcement learning theory developed in the deliverable D1.1.1 and it gives an overview of the API between those two modules.

1 Introduction

This document describes the changes made to the industrial Dialogue Manager to implement the adaptive Reinforcement Learning approach described in deliverable D1.1.1. The Dialogue Manager calls an independent module : the Learning Manager that keeps tracks of the different decisions that have been taken during the dialogue. This document reviews the basic ideas behind the chosen approach to learning in Spoken Dialogue Systems, then it presents the theoretical foundations of the implementation and the API between the Dialogue Manager and the Learning Manager. Finally it discusses the further work planned on statistical Dialogue Management in this area.

2 Reinforcement Learning

2.1 Basic Ideas

The implemented approach is an interesting alternative model to the conventional Markov Decision Process (MDP) approach. While classical reinforcement learning algorithms position their learning at the system action level, the fundamental idea behind this algorithm is to learn at a finer-grained internal decision level. For instance, the response of a dialogue system to a user corresponds to the system action level, but such a system action is actually a combination of several more detailed internal decisions: what to say, what to ask, how to do it, etc. The implemented approach enables each decision point to be optimised independently according to the presumed dependencies of this decision. Thus, this approach reduces the complexity of the system and consequently speeds up the learning. This document presents the implementation of an original framework for learning these internal decisions.

If we refer to the existing automata-based Dialogue Manager, we find an analogy between internal decisions and the transitions between states. Currently these internal transitions are directly handcrafted by the application designer and attempting to learn them all automatically would overload the Learning Manager. Fortunately, however, one does not need to log all the decisions, one only needs to log the decisions that can be controlled. These are the decisions where the application designer specified several alternatives. As a consequence, the Learning Manager will only log the decisions made in a state where the system is supposed to learn, as defined by the designer.

The underlying theory has been extensively described and explained in deliverable D1.1.1. The main concepts in this theory are reviewed in the next section.

2.2 The design architecture

It is supposed that the architecture of the Dialogue Management is an automaton (see figure 1) augmented with a local and global context represented by a set of local and global variables. Hence a state of the world is a state of the automaton with its associated context. Each decision/transition leads to another state and possibly to an update of the context.

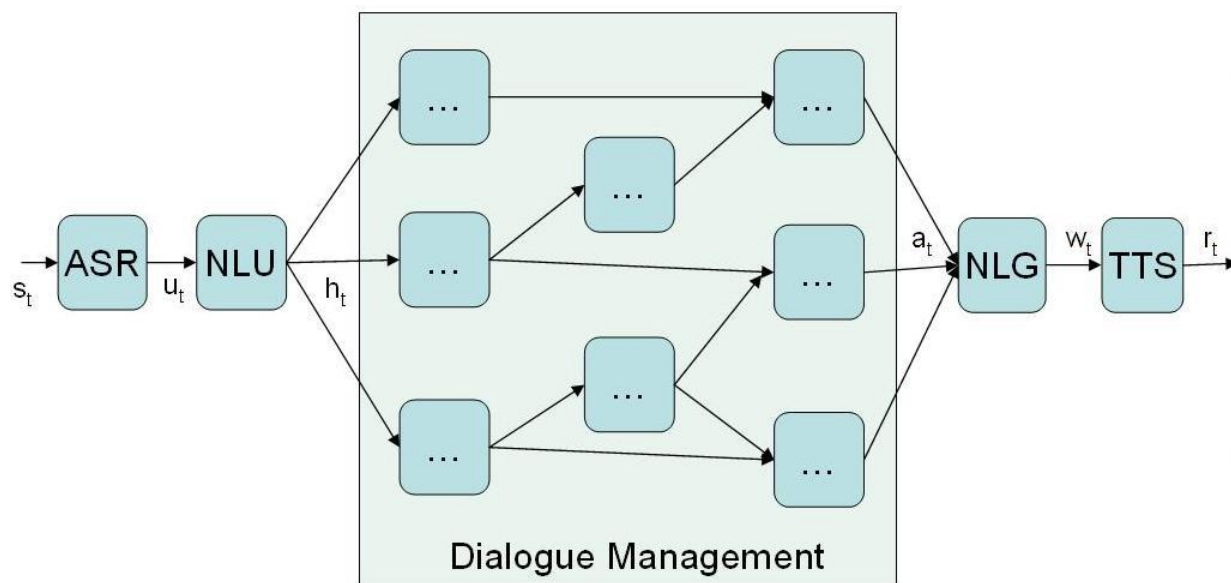


Figure 1: The design architecture.

More formally, a *module* is a processing unit that can choose an *action* (a set of variable instantiations) according to its *context* (a set of instantiated variables). It is equivalent to a state of the design automaton. Every module has a *policy*. The policy governs the decisions that are taken within the module in a given context. A policy is a function from the context space into the action space: $\pi(\text{context}) \mapsto \text{action}$.

A policy makes *decisions*. A decision has the following features: the module where the decision is made, the context affecting the decision and the action which is the choice made by the policy.

$$\text{decision} = (\text{module}, \text{context}, \text{action}) \quad (1)$$

A *dialogue* is a chain of decisions (the user dialogue moves are recorded via the contexts of each decision). Formally, the above components constitute a Module-Variable Decision Process (MVDP) (M, V_M, A_M, R) where:

- M is the set of modules

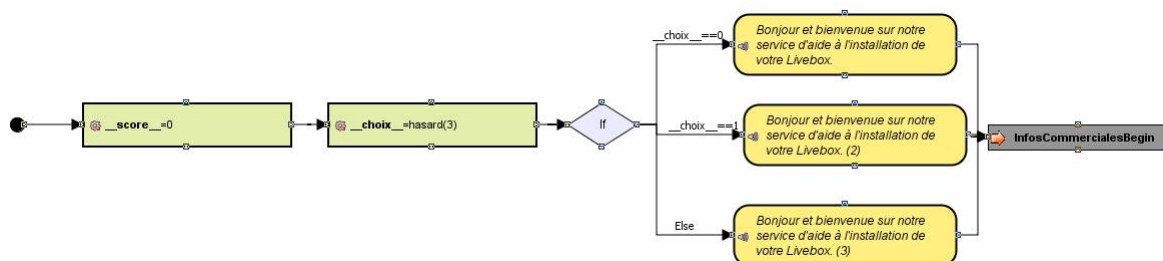


Figure 2: Point of choice mechanism

- V_M is the set of relevant variables (or inputs) in a given module $m \in M$
- A_M is the set of possible actions (or outputs) in a given module $m \in M$
- $R : V_M \mapsto \mathbb{R}$ defines the immediate rewards

This MVDP framework underpins the approach to learning implemented in the D1.1.3 prototype. As an example of its use, figure 2 shows a learning module called a “switch-block”. The set of relevant variables V_m is empty in this case but it could be for instance the age and/or the language register of the user. The three possible actions A_m are shown by the arrows, i.e. the actions are restricted to the choice of the next module. The reward granted by this “switch-block” module would typically be 0. Then, depending on the user reaction to the chosen prompt, the system would access a module-context state which would probably feedback this point of choice with a non-zero reward.

2.3 Application Programming Interface (API)

The Learning Manager does not require a great deal of information from the Dialogue Manager to be able to learn. It only needs to know which actions it may execute in a given state. It can then build its state space from experience as the dialogues progress. The Learning Manager’s job is basically to determine from each set of experiences (or dialogues) what are the internal decisions responsible for the dialogue success or failure. Each time the Dialogue Manager encounters a state where it wishes to let the system take the decision from its experience, it provides the Learning Manager with the current state and the set of actions it can execute. The Dialogue Manager will also have to provide rewards, expressing the dialogue success or failure. During the early stages of learning, the Learning Manager will choose at random, but as state-action pairs are recognised, the Learning Manager will be able to choose the best action in order to reach a dialogue success as fast as possible with low risks, using the methods presented in D1.1.1.

The API of the Learning Manager provides three main callable methods:

- `Action decision(Session, Module, Context, ArrayList<Action>);`
provides the Learning Manager with the current module, context and the possible actions. The Learning Manager returns the chosen action.
- `void reward(Session, double);`
provides a reward (positive or negative) for the current dialogue state.
- `void end_dialogue(Session);`
Informs the Learning Manager that the current dialogue is terminating.

Further API calls are necessary to tune the learning and optimise the various parameters, but these commands are not required during runtime and they are too detailed to merit description in this overview document.

3 Uncertainty Management

The uncertainty management module that was described in deliverable D1.1.1 [1] has been implemented and integrated to the FT dialogue architecture. This module is called the Context Manager. As a general comment, the use of the Context Manager is much more difficult than the use of the Learning Manager. The API is organized into three main categories:

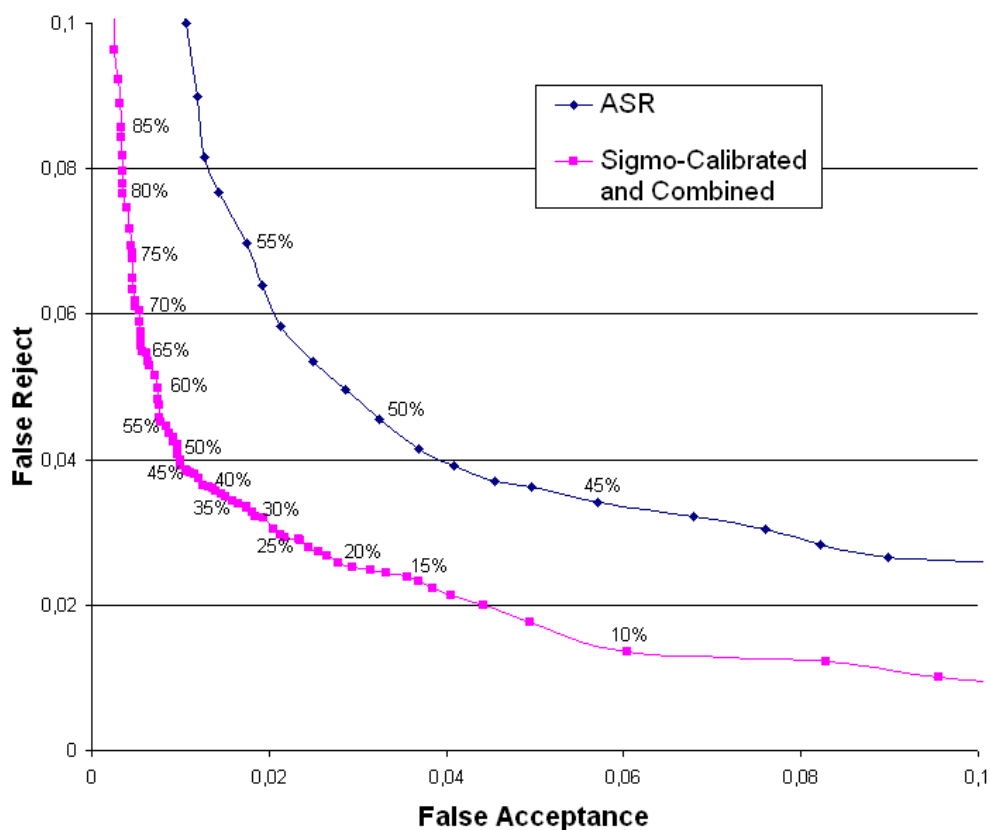


Figure 3: False acceptance versus false rejection.

- The Context Manager creation: all the methods for creating inference rules, parsers, confidence score combination, adding them to the knowledge base, etc.
- The knowledge base modifications: all the methods for adding knowledge, removing information, resetting the knowledge base, etc.
- The knowledge base queries: computes probabilities (inferior, superior and average), getters, high-level queries such as the distribution of knowledge considering a question¹, etc.

This tool has been used in a very basic way in System 4 [2]. It appeared to us that confidence scores that were provided by our Telisma ASR could not be regarded as reliable probabilities. However, after a calibration (learnt function: $[0, 1] \rightarrow [0, 1]$) and a combination (combination rule between the N -best probabilities provided by the LFPR [3, 4]), the probabilities were so reliable that the False Acceptance / False Rejection (FA/FR) trade-off could be divided nearly by two [2].

Figure 3 shows the results we obtained. The blue curve is the FA/FR trade-off given the threshold that has been chosen. The dots on the curves correspond to different values of this threshold. Thus, a 45% threshold provided a FA/FR trade-off remote from a 55% threshold. In addition to providing the system with reliable probabilities, the calibration enabled to move those dots closer and to make the ASR performance/behavior less dependent on the precise threshold value.

¹For instance, if the knowledge base is *ageRomain*, 31 and *ageRomain*, 32 with respectfully the probabilities 0.6 and 0.3, their exists a method to get the full distribution

With calibrated probabilities, we are still facing a second problem: the probabilities of the N -best list items are obtained independently from each other, so there is no guarantee that the sum over these probabilities equals 1. The probabilities we have processed through calibration only inform the probability that the input signal matches each utterance independently from each other. But it does not provide the probabilities that the input signal in fact delivers each utterance. This is where getting “reliable” probabilities matters: we can now combine them in the LFPR. This combination moves the FA/FR curve closer to the origin. The pink curve demonstrates that we drastically improved the overall ASR performance with this method.

4 Remarks

The system with its service (see deliverables D5.3.2 [5] and D5.3.3 [6]) has been deployed and evaluated to generate a corpus of rewards [7, 8]. The reward corpus has been used to train the learning Dialogue Manager built in deliverable D6.3 [9, 10].

The description of task T1.1 was the following: evolution from fully-scripted systems now used in industry towards more robust statistically-based systems. We think that breakthrough advances have been made in this direction:

- The design script has been extended with a policy-based transition between its states. This policy can be controlled randomly, deterministically or with a reinforcement learning algorithm: the Compliance-Based Reinforcement Learning [11, 1, 4].
- The control granted by the industrial process enables us to perform online reinforcement learning without any necessary modification in the algorithm [4].
- We performed one of the first real user experimentations of online reinforcement learning for spoken dialogue systems [9, 10].
- We deployed the first large-scale commercial system with online reinforcement learning [7].
- We developed monitoring functionalities for Key Performance Indicators and reinforcement learning feedback inside the design tool, in order to improve the control of the developers on the learning algorithm [12].
- A new theory for uncertainty management has been developed and implemented: Logical Framework for Probabilistic Reasoning [3, 4].
- The so-built Context Manager has been integrated to System 4 [2].

But there is still a lot of work to do, for instance:

- We need to adapt the CBRL algorithm in an uncertain environment. In other terms, the CBRL algorithm should be implemented in the LFPR.
- The script reading is not made stochastically yet. Only the best path is followed. We have to follow all the paths in order to identify contradictory and reinforcing paths.
- In order to have the CBRL algorithm usable for developers, we have to provide them with tools, such as a convergence predictor, based on the automata structure.
- Sometimes it can be difficult for the developers to design where to give rewards to a dialogue. A way to overcome this issue would be to annotate the first dialogues and apply an inverse reinforcement learning algorithm that learns where to place rewards.

5 Software

The dialogue manager, the context manager and the learning manager are based on the industrial Disserto Suite. The Disserto Suite is based on JAVA for the design tools, and J2EE for the runtime part. The sources are not made available since they are part of France Telecom commercial solution. However, a demonstration of the prototype system will be made available.

References

- [1] R. Laroche, S. Young, O. Lemon, G. Putois, and P. Bretier. D1.1.1: Requirements Analysis and Theory for Statistical Learning Approaches in Automaton-Based Dialogue Management., type = Report, url = <http://www.classic-project.org/>, year = 2009. Technical Report D1.1.1, CLASSIC Project.
- [2] R. Laroche and G. Putois. D5.5: Advanced appointment-scheduling system “system 4”. Report D5.5, CLASSIC Project, 2010.
- [3] R. Laroche, B. Bouchon-Meunier, and P. Bretier. Uncertainty management in dialogue systems. In *Proceedings of the European Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 2008.
- [4] R. Laroche. *Raisonnement sur les incertitudes et apprentissage pour les systèmes de dialogue conventionnels*. PhD thesis, Université Paris VI, France, 2010.
- [5] G. Putois, P. Bretier, and R. Laroche. D5.3.2: Final France Telecom industrial platform adapted to CLASSIC architecture (“System 3”), type = Report, url = <http://www.classic-project.org/>, year = 2010. Technical Report D5.3.2, CLASSIC Project.
- [6] P. Bretier, R. Laroche, and G. Putois. D5.3.4: Industrial Self-Help system (“System 3”) adapted to Final Architecture, type = Report, url = <http://www.classic-project.org/>, year = 2010. Technical Report D5.3.4, CLASSIC Project.
- [7] G. Putois, R. Laroche, and P. Bretier. Online reinforcement learning for spoken dialogue systems: The story of a commercial deployment success. In *Proceedings of SIGDIAL*, Tokyo (Japan), September 2010.
- [8] F. Jurcicek, R. Laroche, G. Putois, and S Young. D1.5: Develop and evaluate training algorithms for both automata-based DMs and POMDP-based DMs which can support on-line adaptation and learning., type = Prototype, url = <http://www.classic-project.org/>, year = 2011. Technical Report D1.5, CLASSIC Project.
- [9] R. Laroche, G. Putois, and P. Bretier. Optimising a handcrafted dialogue system design. In *Proceedings of Interspeech*, Chiba (Japan), September 2010.
- [10] P. Bretier, P. Crook, S. Keizer, R. Laroche, O. Lemon, and Putois G. D6.3: Initial evaluation of TownInfo and Self-Help systems, type = Report, url = <http://www.classic-project.org/>, year = 2009. Technical Report D6.3, CLASSIC Project.
- [11] R. Laroche, G. Putois, P. Bretier, and B. Bouchon-Meunier. Hybridisation of expertise and reinforcement learning in dialogue systems. In *Proceedings of Interspeech. Special Session: Machine Learning for Adaptivity in Spoken Dialogue*, Brighton (United Kingdom), September 2009.
- [12] R. Laroche, P. Bretier, and G. Putois. Enhanced monitoring tools and online dialogue optimisation

merged into a new spoken dialogue system design experience. In *Proceedings of Interspeech*, Chiba (Japan), September 2010.