Towards a Web-based Tool to Support Research Collaboration in Human-Computer Interaction and Cognitive Science*

Antonio Cerone[0000−0003−2691−5279], Anel Mengdigali, Nuray Nabiyeva, and Temirlan Nurbay

Department of Computer Science, School of Engineering and Digital Sciences, Nazarbayev University, Nur-Sultan, Kazakhstan
{antonio.cerone,anel.mengdigali,nuray.nabiyeva,temirlan.nurbay}@nu.edu.kz

Abstract. Human-computer interaction and cognitive science are interdisciplinary areas in which computer scientists and mathematicians often work together with social scientists, such as psychologists and sociologists, as well as with more focussed practitioners such as usability experts and system analysts. In order to work effectively, interdisciplinary teams need to agree on a common communication language as a compromise between the computer scientists and mathematicians' formal modelling approach and the conceptual models normally used by social scientists for describing their domain-related theories and frameworks. Moreover, even when proper communication is established within a specific research team, the next challenge is the presentation of the result to a heterogeneous international community, to allow for cross-fertilisation, exchanges of ideas, work replication and review. This project paper presents the ongoing development of a web-based tool and portal for the modelling and analysis of human cognition and behaviour as well as interactive systems. The aim of the project is for researchers in human-computer interaction and cognitive science to freely use the tool provided by the web portal in order to run in silico experiments, compare the results of in silico experiments and experiments with human beings, perform simulations, analyse systems consisting of computer/physical components and human components, as well as download and upload datasets and models. Domain oriented modelling and visualisation interfaces will ease the modelling and analysis processes by hiding the simulation and formal analysis engines. Finally, the tool will facilitate discussion, review and collaboration. An early prototype of the tool will be available by the end of November 2021.

Keywords: Tool development · Collaborative Research · Interdisciplinary Research · Human-Computer Interaction · Cognitive Science

* Work partly funded by Project SEDS2020004 "Analysis of cognitive properties of interactive systems using model checking", Nazarbayev University, Kazakhstan (Award number: 240919FD3916).

1 Introduction

Research in cognitive science has resulted in the development of a large number of *cognitive architectures* over the last decades [6, 11]. Cognitive architectures are based on three different modelling approaches, *symbolic* (or *cognitivist*), such as Soar [7], which are based on a set of predefined general rules to manipulate symbols, *connectionist* (or *emergent*), such as DAC [13], which count on emergent properties of connected processing components (e.g. nodes of a neural network), and *hybrid*, such as CLARION [12], which combine the two previous approaches.

However, the complexity of these cognitive architectures makes it difficult to fully understand their semantics and requires high expertise in programming them. Moreover, Kotseruba and Tsotsos [6] note that most cognitive architectures have been developed for research purposes rather than for real-life usage. They are actually very specialised tools, each of them only usable within focussed research communities and capable to address one of the following categories of application [6]: psychological experiments, robotics, human performance modelling, human-robot interaction, human-computer interaction, natural language processing, categorisation and clustering, computer vision, games and puzzles, and virtual agents. Finally, although cognitive architectures can mimic many aspects of human behaviour and learning, they never really managed to be easily incorporated in the system and software verification process.

In our previous work, we proposed a notation, the *Behaviour and Reasoning Description Language (BRDL)* [1], for describing human behaviour and reasoning. The semantics of the language is based on a basic model of human memory and memory processes and is adaptable to different cognitive theories. This allows us, on the one hand, to keep the syntax of the language to a minimum, thus making it easy to learn and understand and, on the other hand, to use alternative semantic variations to compare alternative theories of memory and cognition. In our previous work we have implemented parts of BRDL [3, 2, 4] using the Maude rewrite language and system [8, 9]. The automatic translation from BRDL to Maude facilitate modelling, but the results are still the formal textual output from Maude, which is difficult to interpret.

This paper describes a web-based portal and tool that we are developing to address the widespread and heterogenous scientific community that interested in the modelling and analysis of cognitive systems and interactive systems. The portal will allow researchers to collaborate in modelling, compare each other's models, replicate in silico experiments and perform reviewing activities. We exploit BRDL flexibility and extensibility and we use a linguistic approach to define the building blocks of our model, starting from *basic entities*, which are basically names typed on (linguistic) syntactic categories with an associated descriptive definition. Based on Chomsky's concept of *universal grammar* [5], we then combine basic entities using *deep structure* to produce *phrases* [10]. The resultant *structured entities* are then equipped with a semantics that becomes operational when such *semantic entities* are used as components of *dynamic entities*, which are the rules used to model the system. Models are basically sets of dynamic entities. The semantics embedded in the dynamic entities is reflected in the Java

functions that define the simulation engine and the Maude rewrite rules that define the analysis engine.

All defined entities and models are collected in a shared database. The portal allows researchers to access the database, create new entities and models as well as reuse existing entities and reuse and expand existing models, use existing models to run experiments and review models. Our linguistic approach to modelling supports the translation of models into natural language. In this way the output of the simulation engine and the analysis engine become widely understandable for the heterogeneous community of users.

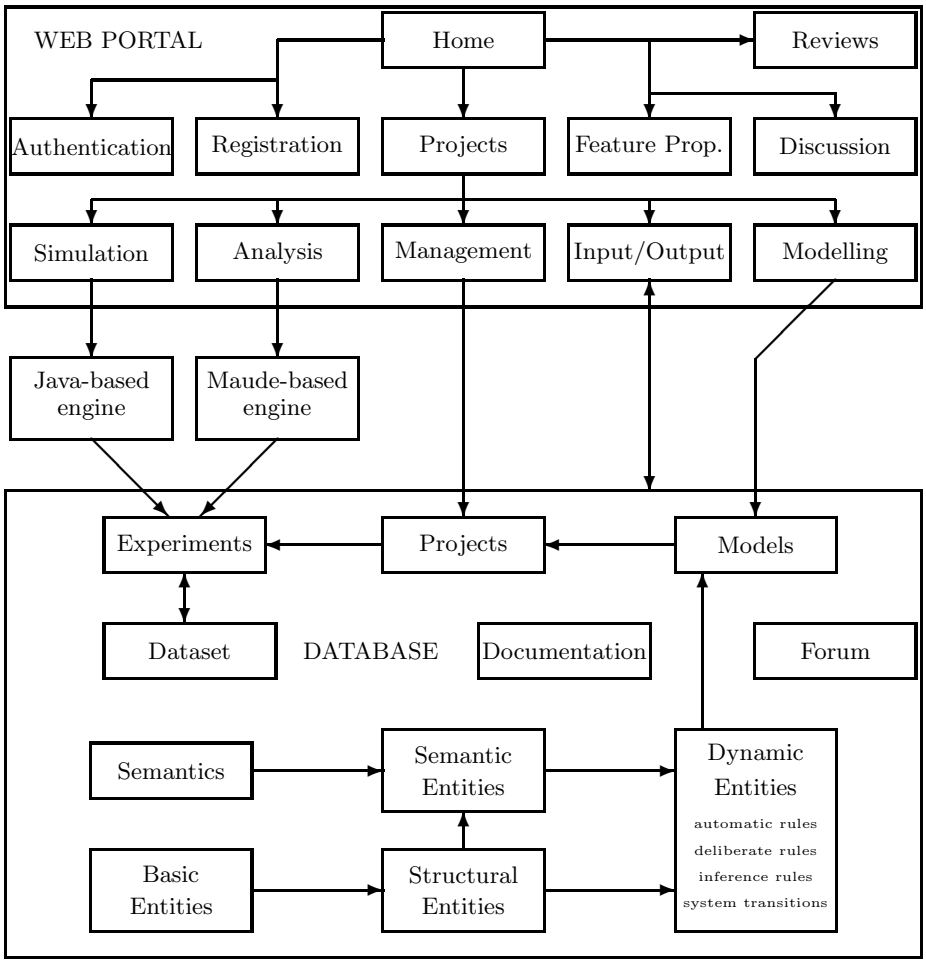

Fig. 1. High-level description of the tool architecture.

The paper is structured as follows. Section 2 informally presents the architecture and functionalities of the tool before delving into the description of the entities that populate the tool database (Section 2.1), the various kinds of models and how their version are managed and used to perform experiments (Section 2.2), and the projects and the stages of their developments the various kinds of models and how their version are managed and used to perform experiments (Section 2.3). Section 3 briefly describes the Java-based simulation engine and the Maude-based analysis engine. Section 4 illustrates the planned web portal and describes its use while Section 4.1 delves into the conversion of a BRDL representation into natural language. Finally, Section 5 presents the tool implementation plan and discusses future work.

2 Architecture and Functionalities

The purpose of the portal and tool is to allow scientists from all over the world to model cognitive and interactive systems, share their models with other researchers and practitioners and collaborate with each other, both within the modelling process itself and by providing feedback and reviews. The tool modelling approach aims at addressing the knowledge domain of a variety of modellers, including computer scientists, interaction designers, usability analysts, cognitive scientists, psychologists and linguists. The current version of the web portal is described in Sect. 2.2. Fig 1 is an informal, high-level description of the tool architecture.

Guests and registered users interact with a web portal that supports the following functionalities:

- user's registration and authentication;
- participation in discussion on the forum;
- management of projects;
- modelling of cognitive and interactive systems;
- model reviews and new feature proposals;
- input and visualisation of datasets;
- document upoload and download;
- experiments performing simulation and analysis of models.

Projects (see details in Sect. 2.3) may be public or private. Any user may access public projects.

Registered users may post messages on the forum and create new projects, within which they carry out the following activities:

1. define basic entities, organise them into structured entities, provide them with a semantics and use them to define dynamic entities (see details in Sect. 2.1), which are the basic components of models;
2. create a new model and become its owner (see details in Sect. 2.2);
3. input datasets;
4. upload documents.

5. modify models;

Registered users are the managers of the projects they create. The managers of a project can admit other registered users as members of that project, assign model ownership and tasks to project members and grant manager role for that project to other members. All members of a project may also carry out activities 1–4 above. The outcomes of these are associated with the user's profile and, if the project is public, become publicly available in the global database and can be used, tested and reviewed by any user. However, they are committed into the project only after the approval of one of the project managers.

All members of a project may also modify models. However, only modifications by model owners or project managers are automatically committed into the project. Modifications by other project members may be committed into the project only after the approval of the model owner or one of the project managers.

In addition all users may

- visualise datasets and view all data associated with public projects and download models from such projects;
- view forum discussions, reviews and feature proposals, though only registered users can post;
- carry out experiments that use models from public projects.

Experiments carried out by non registered users are stored in the database and become publicly available if the user includes a contact email address and this is correctly verified (see details at the end of Sect. 2.2).

2.1 Database Entities

We call *entities* the linguistic constructs that we use as the building blocks of our modelling language and which are close to natural language.

Basic Entities A *basic entity* consists of

name one or more strings of characters, to be used in distinct contexts;
definition a string of characters that is not as general as a dictionary definition but is specific to the abstraction level of the system under analysis and to the world of entities considered;
type which is one of the following linguistic categories:
- *Noun*, consisting of two strings: *singular* and *plural* forms;
- *Instantiation*, consisting of a single string
- *Auxiliary*, consisting of a single string
- *Verb*, consisting of three strings: *simple*, *completed* and *continuous* forms;
- *Attribute*, consisting of two strings: *adjective* and *adverb* forms;
- *Pronoun*, consisting of a single string.

In the following we will use typewriting fonts to indicate names of basic entities (e.g., `animal/animals`) and their structuring, and we will enclose definitions and natural language representations of entities between double quotes. For example, we may have the following basic entities:

- name `animal/animals` of type *Noun*, where `animal` is the *singular* form and `animals` is the *plural* form, with definition: "a living organism that feeds on organic matter and is able to respond to stimuli";
- name `is a` of type *Auxiliary*, with definition: "belong to the upper level of a hierarchy of concepts";
- name `can` of type *Auxiliary*, with definition: "be able to do something";
- name `has` of type *Auxiliary*, with definition: "have as components";
- name `move/moved/moving` of type *Verb*, where `move` is the *simple* form, `moved` is the *completed* form and `moving` is the *continuous* form, with definition (which refers to the *simple* form): "change position according to the direction, speed and time that are provided as arguments".
- name `what` of type *Pronoun*, with definition: "looking for all higher level in the hierarchy".
- name `Lucky` of type *Instantiation*, with definition: "proper name of an individual animal".

A basic entity is defined within a project and, if the project is public, it can be reused by any other project. The same name may be used with different definitions, the same definition may be used with different names, and the same pair name-definition may have different types; each of such triples represents a distinct basic entity.

Structured Entities A *structured entity* consists of a *head*, a number of *arguments* and a *kind*. Head and arguments may be basic entities or structured entities. The kind is one of the following:

- *Noun phrase*, with a *Noun*, a *Pronoun*, an *Instantiation* or a *Noun phrase* as the head and either no arguments or one argument that may be an *Attribute*, a *Noun phrase* or a *value* of any data type;
- *Verb phrase*, with either the *simple* form of a *Verb* or a *Verb phrase* as the head and either no arguments or one argument that may be either the *adverb* form of an *Attribute* or a *Noun phrase*;
- *Participle phrase*, with the *completed* or *continuous* form of a *Verb* or a *Participle phrase* as the head and either no arguments or one argument that is the *adverb* form of an *Attribute* or a *Noun phrase*.
- *Adjective phrase*, with either the *adjective* form of an *Attribute* or an *Adjective phrase* as the head and one argument that is a *Noun phrase*;
- *Adverb phrase*, with either the *adverb* form of an *Attribute* or an *Adverb phrase* as the head and one argument that is a *Participle phrase*;
- *Auxiliary phrase*, which may be *positive* or *negative*, with an *Auxiliary* as the head and two arguments: the first one, called *subject*, is a *Noun phrase*, the second one, called *object*, may in general be a *Noun phrase*, a *Verb phrase* or

an *Adjective phrase*, although some semantic restrictions apply as explained below.

Example of structured entities are:

- *Noun phrases*: `dog`, `what`, `Lucky`, `leg(long)`, `leg(4)`, `leg(long)(4)` and `leg(dog)`;
- *Verb phrases*: `move`, `move(quickly)`, `move(direction(45))(speed(80))`;
- *Participle phrases*: `moved`, `moving`, `moved(quickly)`, `moving(quickly)` and `moving(direction(45))(speed(80))`;
- *Adjective phrases*: `heavy`, `empty`, `full`, `empty(box)`, `empty(box(heavy))` and `empty(heavy)(box)`;
- *Adverb phrases*: `slowly`, `quickly`, `quickly(moved)`, `quickly(moving)` and `slowly(moving(direction(45))`;
- *Auxiliary phrases*: `is a(dog, animal)`, `can(animal(big), move(fast))` `has(dog, leg(4))`, `is a(dog, what)` and `is a(what, animal)` (*positive*) and `is not a(dog, bird`, `can not(tree, move(fast))`, `has not(dog, leg(6))` (*negative*).

Our tool can present some structured entities using natural language. For example:

- "long leg" for `leg(long)`;
- "4 legs" for `leg(4)`;
- "4 long legs" for `leg(long)(4)`;
- "empty box that is heavy" for `empty(box(heavy))`;
- "empty and heavy box" for `empty(heavy)(box)`;

whereas some other structured entities, such as *Auxiliary phrases*, *Verb phrases* and some *Noun phrases* require semantic information to be presented using natural language.

Semantics Entities A structured entity may be given a *semantics* thus yielding a *semantic entity*. This will allow the Java engine to call specific functions and the Maude engine to enable specific rewrite rules whenever the semantic entity is used. For example, a *Noun phrase* may identify a modelled system component and `is a` *Auxiliary phrase* my support the navigation of the hierarchy of concepts and *Verb phrases* built using `move` may support the movement of system components. We have the following kinds of semantic entities:

- *Model Identifier* built on a *Noun phrase*;
- *Fact* built on a *Auxiliary phrase*;
- *Question* built on a *Auxiliary phrase*;
- *State* built on a *Participle phrase*, an *Adjective phrase* or *Adverb phrase*;
- *Action* built on a *Verb phrase*.

Examples of semantic entities are:

- *Model Identifiers* `dog`, `leg(dog)`, `ball` and `John`;

- *State* `empty(box)`, which would change to *State* `full(box)` by performing *Action* `fill(box)`;
- *Action* `pat(dog)`, which may yield *State* `happy(dog)`;
- *Action* `approach(dog(small))`, which may yield *State* `close(dog)`;
- *Action* `move(direction(45))(speed(80))(time(10))`, which may yield *State* `moving(home)`, `moving(office)`, `moved(home)` or `moved(office)`, depending on the initial state;

The given semantics should be documented in the basic entity definition, as it happens for *Actions* that build on *Verb* `move`, which is defined as "change position according the direction, speed and time that are provided as arguments".

Our tool can present semantic entities using natural language. For example, `is a(dog, animal)`, and `is a(dog, what)` are presented in natural language as

- "A dog is an animal." for *Fact* `is a(dog, animal)`;
- "A dog is not a bird." for *Fact* `is not a(dog, animal)`;
- "Is a dog an animal?", for *Question* `is a(dog, animal)`;
- "Is a dog not an animal?" for *Question* `is not a(dog, animal)`;
- "What is dog?" for *Question* `is a(dog, what)`;
- "What is not a dog?",for *Question* `is not a(dog, what)`.
- "Move quickly outdoors the big animal that is indoors" for `move(outdoors)(quickly)(animal(big)(indoors))`;
- "Throw the ball 45 degrees to your left at a speed of 80 km/h" for `throw(direction(45))(speed(80))(ball)`

Semantic entities are associated with constraints in forming structured entities. For example `can` may have as an `object` a *Verb phrase* but not a *Noun phrase*. In fact, 'an animal can move' make sense, but 'an animal can dog' does not.

Dynamic Entities *Dynamic Entities* are the language constructs that describe the dynamics of the modelled system. Human reasoning and behaviour are modelled using

automatic rules $info_1 \uparrow perception \implies action \downarrow info_2$
where $info_1$, $info_2$ may be sets of *Facts*, *Questions*, *States* and *Actions*, *perception* may be a set of *Facts*, *Questions* and *States*, and *action* is an *Action*. *Action action* is triggered by *perception* and performed if $info_1$ is part of the mental state, thus removing $info_1$ from the mental state and adding $info_2$ to it.

implicit attention rules $\uparrow perception \implies \downarrow$ perception
where *perception* may be a set of *Facts*, *Questions* and *States* that is stored as a mental representation.

deliberate rules $goal : info_1 \uparrow \implies action \downarrow info_2$
where $info_1$ and $info_2$ may be sets of *Facts*, *Questions*, *Sttes* and *Actions*, $info_2$ may also include *Goals*, and *action* is an *Action*. *Action action* is triggered by *goal* and performed if $info_1$ is part of the mental state, thus removing $info_1$ from the mental state and adding $info_2$ to it.

explicit attention rules $goal : \uparrow perception \Longrightarrow \quad \downarrow perception$
where *goal* is a *Goals*, and *perception* is a set of *Facts*, *Questions* and *States* that is stored as a mental representationt.

inference rules $info_1 \uparrow \Longrightarrow \quad \downarrow info_2$
where $info_1$ and $info_2$. If $info_1$ is part of the mental state then the inference of $info_2$ from $info_1$ removes $info_1$ from the mental state and adds $info_2$ to it.

System behaviour is modelled using

transition rules $state_1 \stackrel{action}{\longrightarrow} state_2$
where $state_1$ and $state_2$ are sets of *States* and *action* is a set of *Actions*. If the system state contains $state_1$ then the system environment may perform all *Actions* in *action* and adds $state_2$ to the system state.

2.2 Models and Version Control

There are three kinds of models: *cognitive models*, *system models*, *overall models*. Cognitive models and system models are generally called *component models*.

Cognitive Models Cognitive models consist of human memory components and use automatic, deliberate and inference rules to describe human behaviour, reasoning and problem solving. These rules change the mental state defined by the information stored in human memory. How this is achieved is beyond the scope of this paper and can be found in our previous work [1, 3, 2, 4].

System Models System models are essentially described as transition systems. Their evolution is determined by transition rules, which have been introduced at the end of Sect. 2.1.

Overall Models An overall model is the composition of a number of cognitive models and system models. The rules introduced at the end of Sect. 2.1 support the interaction of these models thus yielding the global behaviour of the overall model.

Note that a human being may be modelled by a number of cognitive models that describe responses, mediated by cognition, to mental and physical inputs, and one or more system models, which describe the physical effect of external physical actions. For example, the action of patting the dog and the action of screaming when being hit by a car are produced by the cooperation of cognitive models, whereas the action of falling down when being hit by a car is produced by a system model.

Version Control Each model has a version which is defined using the usual numbering $M.m$, where M indicates major changes and m a minor changes. Versions $0.m$ refer to the requirement analysis and informal specification, before a working models has been built. Models are further categorised into *stable* and *unstable*. Moreover, they may branch out and retain common previous versions, and they may then merge again at some point. For example version 3.7 may branch out as 3.7.a.1.0 and 3.7.b.1.0, and then versions 3.7.a.3.2 and 3.9.b.4.1 may merge as version 3.8 or 4.0, if there are no other branches, or as version 3.7.ab.1.0, if there are other branches. Branching out and merging may also result in a new model, which starts as version 1.0.

Since overall models consist of several component models, every time a new version of a component model is created, the managers of the projects using it are alerted and prompted to create a new version or a new branch for each overall model in order to use the new component model. The managers may do this, or skip this and continue using only the previous version of the component model. The managers may also disable these alerts for a specific component model, possibly enabling them again at some point in the future.

Datasets and Experiments Overall models are used to perform in silico experiments on given datasets and produce new datasets. Experiments may be

- simulation carried out using the Java-based engine;
- model-checking using the Maude-based engine

The experiment setup involves the selection of a specific version of an overall model, possibly the selection of the component models (and their versions) that the selected overall model uses, and a number of quantitative parameters, some cognitive, such as human memory load, human memory access time, cognitive processing times, information decay time, reaction times, and others associated with the performance of computer and physical systems.

The fact that an overall model may be parametric with respect to the component models it uses is an important feature of our tool. It allows researchers to define an overall model of an observable phenomenon and then design experiments referring to alternative theories that explain such a phenomenon. Each theory can be tested by a specific instantiation of the overall model.

Experiments are recorded together with their outcomes and the links to the datasets they used, the versions of the models they used and all related documentation. The tool also aims at providing functionalities that support visualisation of experiment outcomes and statistical information to be used in the comparison of different models.

2.3 Projects and Stages of the Development Process

A project represents the development history of one or more overall models and collects all their versions, documentation and associated experiments. Projects may have various purposes, aiming at collaborative efforts and objectives, such as

- build one specific overall model, which may be used in research or system verification by all users;
- build a number of overall models and compare them with each other and with overall models built within other projects, where the comparison may be carried out in many possible respects, such as the overall model properties or the way it matches real-word data;
- build model components (cognitive models and/or system models) for the benefit of other projects and use the overall models to test such components;
- build overall models to test model components (cognitive models and/or system models) built within other project.

Projects are categorised according to their development stages, inspired by the development stages normally used for open source software products:

Pre-model It is a planning stage and comprises design documents, discussion threads but no models, although models may be under construction at this stage.

Pre-alpha This stage is entered when at least one model is created. During this phase some tentative overall models may also be created and tested. Models as well as the design documents are mostly unstable and subject to frequent changes. In general, models are tested using simple experiments and toy case studies.

Alpha Some models have reached a significant maturity level and may be used in real-world case studies with real datasets. The system architecture may still be changed and details and functionalities are added incrementally. High-level system properties may be verified using model checking.

Beta The system architecture is now stable and includes all major functionalities. Most models are extensively used in real-world case studies with real datasets. The results of in silico experiments and simulations are compared with the results of real-world experiments and the outcome is used to calibrate component models and/or the overall model.

Stable All models are stable and can be now used as research tools to analyse cognitive science theories and to carry out system verification using model checking.

Re-beta Stable overall models are reworked by changing the component models or starting the development or new models. The system architecture is not modified. Testing and using are the same as for the Beta stage

Re-alpha It builds up on a stable version but modifies the system architecture. Testing and usage are the same as for the Alpha stage

The project may go back to the Pre-alpha stage if both system architecture and component models are heavily changed. Versions and development stages are more dynamic and fluid than in software development. It is expected that for a number of projects their stages are continuously 'oscillating' between Stable and Re-alpha/Re-beta. This would be a typical situation when research outcomes are the main goals of the project.

As explained in Sect. 2, models are created within a project. The project owns the model but, if the project is public, any other project may use its models.

This is especially true for component models, whereas overall models tend to be specific to the project and they are only reused in comparative studies. Besides the created models, a project normally includes

- a list of reviews, with links to the specific version reviewed;
- a list of proposed features and extensions to be incorporated in the existing models, with a link to the specific version considered, or to be implemented in a new overall model;
- the experiments performed within the project.

Although these are all visible to guest users, who can also perform experiments, only registered users may write a review or propose features or extensions.

Shared among all public projects are

- a discussion forum, on which any user, guest or registered users, can make comments;
- public datasets, which are initially created by a project but, in terms of usage, may be shared by different projects.
- documentation, which may refer to specific projects, model components, as well as specific datasets or experiments.

Posts and documents are labelled in various ways, depending on their nature, and are linked to projects or model components to which they refer. Private dataset and documentation may be created within private projects and become public if the project is changed to public. Discussions and public datasets and documentation are visible to any user. However, registered users can post on the forum or upload datasets and documentation.

Any project may be deleted by one of its project managers, though the deletion is completed upon the approval of all other managers. Models that are created within the deleted project are preserved only if they are used by other projects. The system administrators will discuss with the manager of such project in order to grant a new ownership.

All uploaded documents will be licensed under one of the Creative Commons (CC) licenses and all models will be licensed under one of the open source licenses. It is still to be decided whether projects will be able to choose specific copyleft licenses.

3 Simulation and Analysis Engines Engines

3.1 Java-based Simulation Engine

Purpose of the Java-based engine is to provide system simulation and presentation of the results. This simulation engine will be integrated into the server-side, which is a Spring Boot project using Java 11 and the Maven framework.

Since an overall system will in general produce a non-deterministic behaviour, it will be possible to run the simulation in several modes: making the choices randomly, stopping at every choice and make the user resolve non-determinism,

use additional models that implement choice strategies and compose them with the overall model. It will also be possible to backtrack to a previous choice and rerun the simulation from there.

3.2 Maude-based Analysis Engine

Real-Time Maude [8, 9] is a formal modeling language and high-performance simulation and model checking tool for distributed real-time systems. Purpose of the Maude-based engine is to provide system analysis using model checking. Models are translated into Real-Time Maude and the rules associated with dynamic entities support the exploration of the state space in order to check system properties.

This approach builds on our previous work [3, 2, 4]. System properties may be expressed either in terms of pattern matching of states to search or in the form of temporal logic properties. The project aims at defining property templates that use natural language to express typical properties of the application domain in order to automatically build the formal specification of the property from standard natural language descriptions.

The Maude code will run in parallel to the backend code on a Linux virtual machine. In this way the web application will have access to the Maude system through bash scripts to ensure consistency of simulation results.

4 Description and Use of the Planned Web Portal

The *Home* page of the portal shows a description of the portal and tool, a list of news, direct links to some top projects and a link to the *Projects* page

The current version of the *Projects* page is illustrated in Fig. 2. Each project is represented by a card showing title, development stage, an illustrative picture and a brief textual description. The card links to the *Specific Project Page*.

The top menu bar allows users to register, login and logout using the *Access* link and to browse the entire database for *Models*, *Entities*, *Dataset*, *Documentation* and *Forum* posts. If a project is selected the *Specific Project Page* is visualised and the top menu bar does not change visually but now refers to *Models*, *Entities*, *Dataset*, *Documentation* and *Forum* posts.

The *Specific Project Page* shows a detailed description of the project, the list of project managers and other members and links to the list of reviews, the list of proposed features and extensions and the experiments performed within the project. Any user may view this information and perform experiments, but only registered users may post reviews and propose features and extensions. Moreover, using the top menu bar any user may access the information related to the project. There is also a link to the *Task Assignment Page* where any registered user can see the task assignment (including model ownership) and project managers can manage tasks.

The *Entities* link opens a page listing all entities used within the project. The *Models* link opens a page listing the models used within the project with links

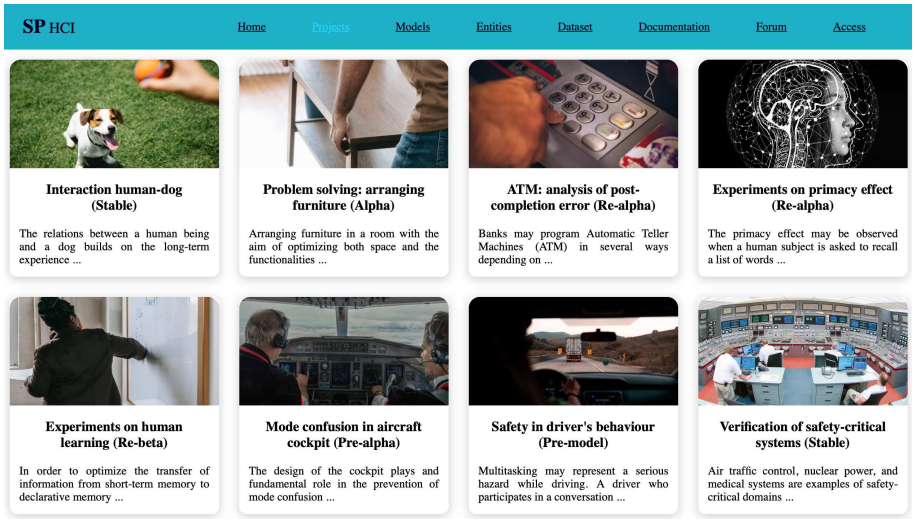

Fig. 2. Project main page.

to the actual versions used within the project by opening a *Specific Model Page*. From here other versions are also accessible. A *Specific Model Page* allows any user to view the model either in BRDL-like form or, as we will see in Sect. 4.1, using natural language.

As we discussed in Sect. 2 project members may also define new entities create new models and edit the existing models using the BRDL-like representation of the model but the changes but only modifications by model owners or project managers are automatically committed into the project.

Domain oriented modelling and visualisation interfaces will be defined to ease the modelling and analysis processes by hiding the simulation and formal analysis engines. In terms of presentation, model and results can be presented to the user in tabular form and, concerning the modification to declarative memory induced by learning processes, also in natural language.

4.1 Presentation in Natural Language

The conversion of a BRDL representation to natural language is driven by the semantics associated with entities and how semantics is realised in dynamic entities. We illustrate presentation in natural language by considering the following example of a cognitive model of a task, in which the subject, who is called John, is willing to pat a dog. The task describes the content of John's declarative memory and is expressed in BRDL as follows:

1. `will(pat(dog))`: $\uparrow$`wagging(dog)` $\Longrightarrow$ $\downarrow$ `wagging(dog)`
2. `will(pat(dog))`: `wagging(dog)` $\uparrow$ $\Longrightarrow$ $\downarrow$ `friendly(dog)`
3. `will(pat(dog))`: `friendly(dog)` $\uparrow$`approach(dog)` $\Longrightarrow$ $\downarrow$ `friendly(dog)`

4. will(pat(dog)): ↑close(enough)(dog) ⟹ ↓ close(enough)(dog)
5. will(pat(dog)): friendly(dog), close(enough)(dog) ↑pat(dog) ⟹ ↓ happy(dog)

Note that sets are represented by separating their elements with commas, but without curly brackets.

The task is presented by the tool in natural language, using templates from the cognitive psychology domain, as follows:

> "Patting the dog is John's goal.
> 1. John's explicit attention focuses on the perception that the dog is wagging.
> 2. From the fact that the dog is wagging John infers that the dog is friendly.
> 3. Since the dog is friendly, John approaches the dog.
> 4. John's explicit attention focuses on the perception that the dog is close enough.
> 5. Since the dog is friendly and close enough, John pat the dog and realizes that the dog is happy."

5 Conclusion and Future Work

We have presented a project that aims at the development of a web portal that allows various categories of scientists and practitioners, including computer scientists, interaction designers, usability analysts, cognitive scientists, psychologists and linguists, to carry out collaborative research in human-computer interaction and cognitive science. Collaboration involves both the modelling process itself and testing and review activities. Cognitive scientists and psychologists may carry out in silico simulations to mimic and accelerate experiments with human subjects as well as long-term learning processes, linguists can perform in silico experiments to emulate human processing of texts and language learning processes, computer scientists may automatically generate formal models and analyses them using the embedded Maude model-checker or exporting them to other analysis tools, interaction designers may model together human tasks and supporting technology and verify properties of the overall model, usability analysts may focus on the verification of usability properties.

An early prototype of the tool will be available by the end of November 2021 It will use the web portal illustrated in Sect. 4 and shown in Fig. 2 and will include the database, the Java-based engine for simulation and the presentation of the results in tabular form as well as natural language. The Maude-based engine and the analysis functionalities will be developed during the first half of 2022. Other functionalities scheduled for 2022 are property templates, graphical and domain-specific visualisation of results and model comparison.

Although our web-based tool address research collaboration in human-computer interaction and cognitive science, the same approach may be used in other application domains. Other possible application domains are coordination model, socio-technical system, systems biology and ecology.

References

1. Cerone, A.: Behaviour and reasoning description language (BRDL). In: SEFM 2019 Collocated Workshops (CIFMA), Lecture Notes in Computer Science, vol. 12226, pp. 137–153. Springer (2020)
2. Cerone, A., Murzagaliyeva, D.: Information retrieval from semantic memory: BRDL-based knowledge representation and Maude-based computer emulation. In: SEFM 2020 Collocated Workshops (CIFMA), Lecture Notes in Computer Science, vol. 12524, pp. 150–165. Springer (2021)
3. Cerone, A., Ölveczky, P.C.: Modelling human reasoning in practical behavioural contexts using Real-Time Maude. In: FM'19 Collocated Workshops - Part I (FMIS), Lecture Notes in Computer Science, vol. 12232, pp. 424–442. Springer (2020)
4. Cerone, A., Pluck, G.: A formal model for emulating the generation of human knowledge in semantic memory. In: Proc. of DataMod 2020, Lecture Notes in Computer Science, vol. 12611, pp. 104–122. Springer (2021)
5. Chomsky, N.: Language and Mind. Cambridge University Press (2006)
6. Kotseruba, I., Tsotsos, J.K.: 40 years of cognitive architectures: core cognitive abilities and practical applications. Artificial Intelligence Review **https://doi.org/10.1007/s10462-018-9646-y** (2018)
7. Laird, J.A.: The Soar Cognitive Architecture. MIT Press (2012)
8. Ölveczky, P.C.: Real-time maude and its applications. In: Proc. of WRLA 2014, Lecture Notes in Computer Science, vol. 8663, pp. 42–79. Springer (2001)
9. Ölveczky, P.C.: Designing Reliable Distributed Systems. Undergraduate Topics in Computer Science, Springer (2017)
10. Pinker, S.: The Language Instinct. William Morrow (1994)
11. Samsonovich, A.V.: Towards a unified catalog of implemented cognitive architectures. In: Biologically Inspired Cognitive Architectures (BICA 2010), pp. 195–244. IOS Press (2010)
12. Sun, R., Slusarz, P., Terry, C.: The interaction of the explicit and implicit in skill learning: A dual-process approach. Psychological Review **112**, 159–192 (2005)
13. Verschure, P.: Distributed adaptive control: A theory of the mind, brain, body nexus. Biologically Inspired Cognitive Architectures **1**, 55–72 (2012)