Amine Platform: an Artificial Intelligence Environment  
For the Development of Intelligent Systems  

A. Kabbaj\textsuperscript{1}, K. Bouzouba\textsuperscript{2}, A. Soudi\textsuperscript{3}  

\textsuperscript{1} INSEA, B-P 6217, Rabat, Morocco  
\textsuperscript{2} EMI, Mohamed V University, Av. Ibn Sina, B-P 765, Rabat, Morocco  
\textsuperscript{3} ENIM, Av. Hadj Ahmed Cherkaoui, B-P 753, Agdal, Rabat, Morocco  

Emails: akabbaj@insea.ac.ma, karim.bouzoubaa@emi.ac.ma, asoudi@enim.ac.ma  

Abstract. This paper presents an overview of Amine; a multi-layer and open-source\textsuperscript{1} platform, implemented in Java and dedicated to the development of intelligent systems and agents. Amine is composed of four layers: a) a kernel layer that enables the creation, edition, update and manipulation of multi-lingua ontologies, b) an algebraic layer that offers a set of elementary data types (AmineInteger, AmineDouble, etc.) and structured types (AmineSet, AmineList, Term, Concept, Relation, Conceptual Graph) and various matching-based operations (like match, equal, unify, subsume, maximalJoin, generalize, analogy, compare, etc.), c) a programming layer that provides three programming paradigms: i) an ontology or memory-based programming paradigm which is concerned by incremental and automatic integration of knowledge in an ontology (or agent memory) and by information retrieval and other related ontology/memory-based processes, ii) a pattern-matching and rule-based programming paradigm, embedded in PROLOG+CG language which is an object based and CG-based extension of PROLOG language, and iii) an activation and propagation-based programming paradigm, embedded in SYNERGY language, and d) an agent and multi-agent systems layer that enables the development of agent-based applications.  

1. Introduction  
Amine is a multi-layer and open-source\textsuperscript{1} platform, implemented in Java and dedicated to the development of intelligent systems and agents. Amine is composed of four layers:  

1. Kernel layer: It concerns the creation and manipulation of multi-lingua ontology. Amine offers a possibility to create, edit and ask an ontology that is defined in terms of Conceptual Structures (definition, canon, individual, situation and context). Several conceptual lexicons can be associated to a conceptual ontology (i.e. lexicons for English, French, Spanish, etc.) allowing the creation and use of multi-lingua conceptual ontology. Users, with different languages, can express knowledge in their languages and still share all knowledge expressed in other languages and integrated in the same ontology.  

2. Algebraic layer: It provides several types of structures and operations: elementary data types (AmineInteger, AmineDouble, etc.) and composed types (AmineSet, AmineList, Term, Concept, Relation and Conceptual Graph –CG–). It provides also various matching-based operations (like match, equal, unify, subsume, compare, maximalJoin, generalize, analogy, etc.).  

3. Programming layer: Three complementary programming paradigms are provided by Amine: a) an ontology or memory-based programming paradigm which is concerned by incremental and automatic integration of knowledge in an ontology (or agent memory) and by information retrieval and other related ontology/memory-based processes, b) a pattern-matching and rule-based programming paradigm, embedded in PROLOG+CG language which is an object based and CG-based extension of PROLOG language and, c) an activation and propagation-based programming paradigm, embedded in SYNERGY language.  

4. Agents and Multi-Agents Systems layer: We are using available Java Agent Development Environments (like JADE\textsuperscript{2}) to explore the use of Amine components in the development of agent-based applications.  

Amine provides also several graphical user interfaces (GUIs), a web site, several samples and a growing documentation and applications. Figure 1 presents “Amine Suite Panel” which provides an access to all GUIs of Amine (Lexicons_Ontology editor GUI, CG Notations editors GUI, CG Operations GUI, Dynamic Ontology GUI, Ontology processes GUI,  

\textsuperscript{1} http://sourceforge.net/projects/amine-platform  
\textsuperscript{2} jade.tilab.com/
Prolog+CG GUI, Synergy GUI), as well as an access to some ontology examples and to several tests that illustrate the use of Amine structures and their APIs.

Amine four layers form a hierarchy: each layer is built on top of and uses the lower layers. However, a lower layer can be used by itself without the higher layers: kernel layer can be used directly in any application, algebraic layer can be used directly too, etc. Among the goals (and constraints) that have influenced the design and implementation of Amine was the goal to achieve a higher level of modularity and independence between Amine components.

The paper is organized as follows: the second section introduces briefly the kernel layer. The third section presents the algebraic layer. The fourth section presents the new version of Prolog+CG. Dynamic Ontology engine is described in (Kabbaj et al., 2005) and Synergy language is described in a previous paper (Kabbaj, 1999) and in the web site of Amine. The fifth section describes briefly our approach concerning the development of agent-based applications.

We urge the reader to consult the web site of Amine for more detail. The source code and the documentation can be downloaded from sourceforge site.

2. Amine Kernel layer
The Kernel layer of Amine enables the creation and manipulation of multi-lingua ontology. Ontology topic is a multi-disciplinary subject, treated for centuries by philosophers, linguists, cognitive psychologists, researchers in Artificial Intelligence, cognitive scientists, and now researchers in semantic web. John Sowa provides a good synthesis in his second book (Sowa, 2000): "The subject of ontology is the study of the categories of things that exist or may exist in some domain. The product of such a study, called an ontology, is a catalog of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language L for the purpose of talking about D. The types in the ontology represent the predicates, word senses, or concept and relation types of the language L when used to discuss topics in the domain D." This "catalog of types" is organized, in general, in a hierarchy, called type hierarchy.

This definition of an ontology can be extended by joining to each type all the knowledge acquired by the system concerning this type. Such knowledge can be organized in terms of Conceptual Structures (CSs): type definition, canon for a type and schemata associated to a type. Individuals (instances) can be associated to their types too. Amine offers a possibility to create, edit, update and use ontologies according to the above extension. Thus, an ontology in Amine is a graph of nodes that represent Conceptual Structures (CSs). Actually, five types of nodes are defined:

- **Type node**: a node that represents a type (a category). This node can contain the definition of the type and/or the canon of the type. Relation types are considered as types and are represented by RelationType nodes (a specialization of Type node).
- **Individual node**: a node that represents an individual (an instance). This node can contain the description of the individual.
- **Situation node**: a node that contains the description of a situation. A situation can be specific or generic, and it can be indexed under several types and/or individuals.
- **Context node**: a node that contains the description of a context. This type of nodes is used for the representation and integration of compound descriptions (like compound CGs).
- **Metaphor node**: a node that represents a metaphor. See Amine Web Site for more detail.

Figure 2 provides a snapshot of an ontology edited with LexiconsOntology GUI. The ontology

---

3 http://amine-platform.sourceforge.net

4 http://sourceforge.net/projects/amine-platform
is visualized in the main frame as a hierarchy. Tree nodes represent CSs nodes of the ontology (each type of node is represented by a different color). A Type can be specialized by other types (and related to them by the (s)pecialization link), it can be related to several individuals (by (i)ndividual link) and it can be related to several situations (by (u)tilisation link). The content of a node is visualized, if required, in an auxiliary frame.

Figure 2: An example of Ontology in Amine

3. Amine Algebraic layer

The algebraic layer provides various types of structures and operations. In addition to elementary Java objects (Boolean, Integer, Double and String) and Amine proper elementary objects, Amine provides collection structures (like AmineList which extends the Java class ArrayList and AmineSet which extends the Java class HashSet) and complex structures: Term, Concept, Relation and Conceptual Graph.

Amine Structures and operations

Amine structures are AmineList, AmineSet, Term, Concept, Relation and Conceptual Graph (CG). Concepts and relations compose a CG, but they may be used also as a "stand-alone" structures. An element in Amine structures is any elementary object or any object that implements AmineObject interface. All Amine structures implements AmineObject and Matching interfaces and they have their own APIs with a set of methods (operations) specific to each structure. A detailed description of these strutures and the associated operations, with examples, is provided in Amine Web Site.

- AmineList extends the Java class ArrayList. Like lists in Lisp and Prolog, the constructor operator is defined on AmineList.
- AmineSet extends the Java class HashSet. Only Amine elementary objects can be elements of an AmineSet.
- A concept is a (Java) object composed of a type, a designator with an optional coreferent, a descriptor, a state (used in the context of Synergy language) and a point (used in the case of graphical display of the concept). Only concept type is mandatory, the other components of a concept are optional.

Concept type is either a variable or a reference to a Type CS contained in the current ontology. Concept designator is either null (i.e. the designator is not specified), a variable (with, by default, an existential quantifier), a reference to an Individual CS contained in the current ontology, a set of Individuals, or a pseudo-designator. Concept coreferent, if specified, is represented by a variable shared by concepts that belong to different contexts. For instance, the two concepts [Human: khalid *x] and [Man ?x] can be two concepts in co-reference, with x the variable that specifies this coreference.

Concept descriptor is either null (i.e. it is not specified), a variable, a reference to a CS or any Amine Object (including CG). If a concept is used to compose Conceptual Graph (CG) structure, the concept can be related, by income and/or outcome relations, to other concepts. Note however that a concept can be used "alone", like any other Amine structure.

- A conceptual relation relates a concept to another concept. It is composed of a type, a source (concept), a target (concept), a collection of points (for the segments of the relation) and the position of the type identifier (if relation drawing is considered). Relation type, source and target concepts are mandatory. Relation type is a RelationType CS contained in the current ontology. RelationType can be a variable. Alone, a conceptual relation can be used as a dyadic predicate.

- A Conceptual Graph (CG) is a directed graph of nodes that correspond to concepts, connected by labeled and oriented arcs that represent conceptual relations. In Amine, only dyadic relations are considered. This constraint has been adopted since most of the relations are dyadic.
and the implementation is more efficient. Monadic and more than dyadic relations can be expressed in term of dyadic relations. Of course, this constraint can be eliminated if necessary.

While Amine allows the formulation of general CG, it is strongly recommended to formulate CG as functional CG, especially when CGs are used in an application that uses matching-based CG operations. Functional CG is a CG in which income relations types of each concept are mutually different, and the same constraint should hold for outcome relations types of each concept. Matching based CG operations in Amine are defined on functional CG but the other operations of CG API are defined on general CG (being functional or not). The rationale for this restriction on matching based CG operations is discussed in Amine Web Site.

**CG notations**

Amine platform supports three CG notations (Linear, CGIF and Graphic notations). At the front side, a user can enter his CGs in any one of three notations, Amine parses the notation and produces an internal representation (a Java object) for the CG. At the end side, Amine can formulate the internal representation of a CG in LF, CGIF, or Graphic notation. As a consequence of this flexibility in CG notations, a user can provide, for instance, a CG in LF and ask Amine to return the same CG in CGIF or in Graphic notation.

Amine provides CGNotations GUI: a multi-lingua and multi-notations CG editors GUI. A CG can be expressed in different languages of the current ontology and in different notations. Figure 3 presents the same CG in the three notations.

(a) a CG in LF notation

```
[Begin]-
-obj->[Session],
-agnt->[Person : John]
-srcs->
[Proposition = [Press]-
-obj->[Key : enter]-partOf->[Keyboard],
-agnt->[Person : John] ]
```

(b) The same CG in CGIF

```
[Begin 0]
(srcs 0
 [Proposition=[Press #1][Key :enter]
 (partOf enter [Keyboard])
 (agt #1 [Human :John])
 (obj #1 enter) ])
(agt 0 [Human:John])
(ob 0 [Session])
```

(c) The same CG in Graphic notation

**CG Operations**

Apart from operations specified in AmineObject and Matching interfaces and implemented by CG class, CG has a large set of specific operations: getters and finders to get and find concepts and/or relations, checkers to check various constraints, adders and removers to add/remove concepts and/or relations, setters to set values for specific parameters, file operations (Load/Save) to load/save a CG from/in a file, I/O operations to read/write a CG according to the different CG notations, etc.

Also, in addition to the operations specified in Matching interface and implemented by CG class (match(), equal(), unify(), subsume(), maximalJoin(), and generalize()), there is a set of specific CG operations that are either variants or derived from the Matching operations: specialize(), isCanonic, expand(), contract(), coveredBy(), analogy(), compare(). A detailed description of these operations, with examples, is provided in Amine Web Site.

**CG Operations GUI**

To familiarize users with matching based CG operations and to enable a quick testbed for these operations, Amine provides a CGOperations GUI. This GUI provides also a possibility to see an animation of the operation, i.e. a possibility to see
the application of the operation step by step on the drawing of the two CG; to see how the progressive matching of concepts and relations produce the new CG that results from the matching.

Figure 4 illustrates the use of CGOperations GUI. It shows the classical example of analogy: analogy between solar system (the source) and atomic system (the target) (Falkenhainer et al., 1990). targetCG before the analogy is specified in the left-top panel. In the right-top panel, the sourceCG is provided. The bottom panel shows the result of the analogy operation: the new description of the targetCG (after the analogy).

4. Re-engineering and integration of Prolog+CG in Amine

Previous versions of Prolog+CG have been presented in (Kabbaj and Moulin, 2005) (Kabbaj et al., 1990). Pr. Peter Ohrstrom, Henrik Scharfe and their team developed a very good on-line course on some aspects of Prolog+CG. Let us recall three key features of Prolog+CG:

- CG (simple and compound CGs) is a basic and primitive structure in Prolog+CG, like list and term. And like a term, a CG can be used as a structure and/or as a representation of a goal. CG matching-based operations are provided as primitive operations.
- By a supplementary indexation mechanism of rules, Prolog+CG offers an object-based extension of Prolog.
- Prolog+CG offers an interface with Java: Java objects can be created and methods can be called from a Prolog+CG program. Also, Prolog+CG can be activated from Java classes.

The three key features are still present in the new version of Prolog+CG but the re-engineering of Prolog+CG 2.0, which was necessary for its integration in Amine platform, involved some changes in the language. Four main changes are of interest:

- Type hierarchy and Conceptual Structures (CSs) are no more described in a Prolog+CG program. Now, Prolog+CG programs are interpreted according to the current ontology. Type hierarchy and CSs can be looked for directly from the ontology. Also, Prolog+CG attempts first to interpret each identifier in a program according to the current lexicon of the current ontology. If no such identifier is found, then the identifier is considered as a simple identifier (without any underlying semantic).
- The notion of project is introduced: user can consult several programs (not only one) that share the same ontology. For instance, Figure 5 gives a snapshot of a natural language application composed of four programs (lexicon, Analysis, Utilities, QuestionAnswering).
- Prolog+CG inherits the first two layers of Amine: all Amine structures and operations are also Prolog+CG structures and operations.
- The interface between Prolog+CG and Java is more simple and more "natural": the call of a method in Prolog+CG is very similar to the call in Java. Here is some examples:

```
http://www.hum.aau.dk/~scharfe/vid/lecture.htm
```
5. The Agents and Multi-Agent Systems layer

Instead of developing, starting from scratch, our own Java agent development environment, we decided to use available open-source environments and to integrate them in Amine in order to explore various possibilities that can result from this integration. For instance, we are exploring with ElHari 6 the use of JADE 7, a Java open-source agent development environment, which offers possibilities to create agents with different kinds of behavior and with communication capabilities. The first application developed by ElHari was the use of JADE and Amine in order to simulate a tutoring situation where students interact with their teacher. Students ask questions and the teacher responds. The teacher has an access to an ontology and "knows" what primitives (methods from the kernel APIs) to use to respond to specific questions. Figure 6 gives a snapshot of this simple application: the student Andre asks the teacher Socrate to give him the subtypes of Human in English and the student Menon asks Socrate to give him the definition of the type Man in French.

![Figure 6: Agent-based application using Amine Platform](image)

Currently, we are working on more complex agent-based applications, with intentional capabilities (knowledge, goals, plans, believe, etc.) and more complex behaviours (reasoning, planning, experience-based processing, etc.).

6. Conclusion

This paper provides an overview of Amine: an open source platform, implemented in Java and dedicated to the development of intelligent systems and agents. Amine has a multi-layer architecture, composed of four layers and several GUIs: ontology, algebraic, programming and agent layers. These layers form a hierarchy: each layer is built on top of and use the lower layers. However, a lower layer can be used by itself without the higher layers.

References


Kabbaj A., Synergy as an Hybrid Object-Oriented Conceptual Graph Language, in Proc. Of the 7th International Conference on Conceptual Structures (ICCS’99), Springer-Verlag, 1999.


---

6 In the context of her DESA intitled « The use of Amine in the development of agent-based applications »
7 jade.tilab.com