Ontology-driven Agent Code Generation for Home Care in Protégé

Akos Hajnal, Gianfranco Pedone, Laszlo Varga

Computer and Automation Research Institute of the Hungarian Academy of Sciences, Kende u. 13-17, 1111 Budapest, Hungary
{ahajnal, gianfranco.pedone, laszlo.varga}@sztaki.hu

1 Introduction

The percentage of old and chronically ill people in all European countries has been growing steadily in the last years. This tendency is putting a very heavy economic and social pressure on all national Health Care systems. It is widely accepted that this problem can be somehow palliated if *Home Care* (HC) services are improved and used as a valid alternative to hospitalisation. The aim of the *K4Care* European project is to explore the use of IST and Artificial Intelligence techniques to provide a *Home Care model*, design and develop a prototype system, based on Web technology and intelligent agents. Our presentation concerns the realization of a plug-in for Protégé that automates the generation of the entire Multi-Agent System's code, underlying the application domain, starting from a complete ontological description of the environment. This plug-in is intended for designers and developers of K4Care platform, giving them the possibility, among the others, to supervise the state of the application ontologies, to modify them, to generate the inherent agent-oriented code, to embed the control of the MAS middleware (in our case, JADE), to register created agents in the MAS and start them. The plug-in is an integrated tool enabling agents generation in a knowledge intensive domain, such as health care. In our presentation we will introduce the development of all ontologies involved in agents generation, the theoretical approach to the plug-in realization and the technical details related to its development.

2 K4Care application ontologies

The Home Care model created during the project defines actors in the care model (e.g. physicians, nurses, social workers, patients), their roles and interactions. Concepts of the model are formalized using ontologies that are a standard AI knowledge representation mechanism. The language for ontologies is chosen to be the OWL (Web Ontology Language) from the World Wide Web Consortium (W3C), and we used Protégé-OWL plug-in to create and manage OWL ontologies. They represent the knowledge assets of the K4Care application and the catalyst for the agents behavioural model, as well as the fundaments for the agents code generation. The domain knowledge description is separated from the software realization, assuring a high level of interoperability and independence among elements of the system.
We have divided *application ontologies* into different sources, in relation to their application coherence. All the ontologies are independent of each other, except of the *K4Care* global one, which includes all the others and contains necessary cross-references to relate concepts coming from different elementary ontologies. The application ontologies in K4Care are:

- **Domain ontology**, divided into *Actor Profile Ontology* (APO) and *patient-Case Profile Ontology* (CPO). APO represents the profiles of the subjects in *K4Care* model (healthcare professionals, patients and social organisms) and contains the skills, concerns, aspirations of the people that they represent, together with the healthcare services that those people offer to or receive from the *K4Care* model. CPO represents the relevant medical concepts (e.g. symptoms, diseases, syndromes). Domain ontologies describe "know-what"

knowledge about actors accessing the *K4Care* model and pathologies the *K4Care* model gives support to.

- **FIPA ontology**. The general conceptualization of a MAS development standardization. It represents "know-what" knowledge of a MAS (messages, encodings, communication languages, and so on) and "know-how" knowledge of their interactions (communicative acts, interaction protocols, communication ontologies, and so forth).
- **GAIA ontology**. It includes the ontological description of the MAS development methodology adopted within the *K4Care* project, with particular attention to concepts such as *role*, *responsibility*, *permission*.
- **JADE ontology**. It expresses the implementation and deployment concepts that must comply with the elected MAS of the project, i.e. JADE (agent-types, behaviours, containers, mobility, and so forth).
- ***K4Care* ontology**. This ontology was required in order to model the inevitable ontology-cross references: the agent capabilities, for example, are expressed in terms of "*actions*" in the APO; these are considered as "*responsibilities*" or "*permissions*" by GAIA; the behavioural logic of an agent inherent to its capabilities is expressed in JADE in terms of "*behaviours*", which can contain FIPA compliant "*interaction protocols*", as well.

3 Agent Code Generation

In this project we automatically generated agent-oriented code from the application ontology. The accuracy and the level of details of the code is directly proportional to the formalism adopted to describe the domain model. Particularly important is the representation of the agent capabilities, which should consist of a formal representation of their business logic. The objective of this automation process was to analyze, recognise and extract all the MAS compliant agent-oriented information: we had to take into account, first of all, the implementation requirements deriving from the elected MAS structure (agent class definition, protocols, message structures, message contents, behaviours class). In order to guarantee flexibility in the ontological traversing and parsing, and to decouple the conceptualization layer from the parsing one, the introduction of a mapping dictionary was necessary. This thesaurus described to the parser the "root" level of concepts where to start collecting agents knowledge from. All relevant extracted information has been temporally represented in an internal model. We have obtained agents code combining together two categories of elements: *fixed and invariable* (keywords related to programming language, JAVA, and the MAS, JADE), and *knowledge-model-derived*, necessary to embed expressions which comply with the correct code completion of an agent. The agent's capabilities (domain knowledge) are represented by its actions, listed in the APO: from the paradigm's point of view, these must be translated in terms of behavioural models accepted by the MAS. The code was built taking into account the following aspect: different behaviours have been dedicated to different skills in order to interact in parallel with different other entities (Execution Test Engines). This is a fundamental aspect for the JADE agents: even if behaviours can be of different nature (parallel, cyclic, one-shot, and so on), the execution of one behaviour inhibits (blocks) the execution of all the others (following the policy of a "behaviours queue"). We propose a solution to this problem while generating the agent code: we instantiate another same behaviour at the very beginning of the invocated one, in order to grant real parallel interaction invocations of the same nature (a sort of "*dynamic behavioural pooling*"). Most of the effort in the agent code generation was also dedicated to obtain a harmonious code arrangement of all concepts (elements) characterizing the domain model and the application development.

4 K4Care platform MAS generation Plug-in

The platform of the prototype system is chosen to be JADE (Java Agent DEvelopment Framework). JADE is a FIPA compliant middleware that supports the implementation of multi-agent systems, that is. The K4Care agent code generator is implemented as a Protégé plug-in. In

Protégé, the OWL domain ontologies can be opened, viewed and edited. The plug-in is a tab widget, thus the code generator tab can be added as a new tab in the opened K4Care project ontology. The plug-in, in this way, accesses domain ontologies via Protégé's OWL API, and shows a swing-based user interface with text fields to specify deployment parameters (JADE home, JAVA home, output directories, output package names and so forth) and facilities to generate and compile agents source files. The plug-in, in addition to the previous features, is able to start JADE agent platform, launch test agents for each actor type (the number of agents is parameterized), and one (or several) test engines that invoke random actions of randomly selected agents in specified time intervals. This latter function is be used to benchmark, stress test the current configuration (hardware, software), measure performance, response times of agents at different loads. The generated agent codes consist of a set of Java classes. These classes can be grouped into three main categories: behaviour classes, message ontology classes, and agent classes. Behaviour classes implement different actions of agents. Message ontology defines the ontology of elements that agents can use within content of messages. Agent classes implement different agent types that start the behaviours and register the related (message) ontologies. The code generator relies on the application ontologies presented in the previous sections, and also on an implementation of different actions declared in the APO. There are several alternatives to define actions at different abstraction levels (e.g. using business process description language), the plug-in currently expects a Java library in which actions are implemented by simple Java methods. Method names are given by the action names in the domain ontology so that they can be derived by the code generator. Complex actions based on specific protocols (that, for example, require interaction with other agents within action body) are implemented by several methods invoked at specific states of the protocol. Activities of agents are embedded in agent behaviours. Possible activities of agents are the actions, therefore behaviour classes can be generated by iterating through all the actions in the APO. For each action a unique behaviour class is created that extends a behaviour schema corresponding to the protocol of the action (as declared in the domain ontology). For example, behaviours following FIPA-Request protocol extend JADE's built-in *AchieveREResponder* class (Achieve Rational-Effect Responder) that implements the FIPA-Request protocol. Depending on the given protocol the related call-back methods must be overridden to perform specific activities at specific states. These methods are generated by the plug-in with body containing the appropriate method calls to the external library of actions. The proper communicative acts of messages, the conversation-id, protocol, ontology fields are automatically set by the generated code according to the given protocol. By default, a given active (ongoing) behaviour cannot be started again until its protocol finishes. To permit parallel activities of the same behaviour in the same agent each behaviour is created so that on request register first a new (listener) behaviour if all the behaviours are active currently, then start the protocol of the activity. This way, there will always be at least one listener behaviour to fulfil a new request, and the behaviour pool dynamically increases according to the number parallel requests for the same behaviour. Message ontology classes define the concepts that can be used in messages exchanged between agents. According to FIPA standards, request messages contain agent actions (*AgentActionSchema* in JADE), while result messages contain predicates (*PredicateSchema*). Agent actions and predicates can contain slots (fields) as well as substructures defined as concepts (*ConceptSchema*). JADE ontologies consist of a set of Java classes (with the appropriate schema superclass), and ontology classes that register these elements in JADE. All these classes are generated by the plug-in using the domain ontology.

Actors are represented by agents in the system, but the code generator creates one agent class for a given actor type (the particular actor is a start-up parameter). The capabilities of agents are determined by the actions declared for the actor type in the APO. Behaviours are registered with the appropriate message template: communicative act, protocol, ontology, that is, behaviours can only be activated by a dedicated message belonging to the action. In addition, agents are capable of handling unknown messages sent to the agent (logged), and administer its life-cycle (suspend, activate, takeDown, save and recover agent state, etc.).