Component Agent Framework for domain-Experts (CAFnE) Toolkit

Gaya Jayatilleke John Thangarajah Lin Padgham Michael Winikoff

RMIT University
Melbourne, AUSTRALIA
{gjayatil,johthan,linpa,winikoff}@cs.rmit.edu.au

ABSTRACT

The Component Agent Framework for domain-Experts (CAFnE) toolkit is an extension to the Prometheus Design Tool (PDT). It uses the detailed design produced by PDT with further annotations by domain experts to automatically generate executable code into a desired agent platform. The key feature of CAFnE is that it allows domain experts with limited programming skills to easily build and modify agent systems.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments

General Terms

Design, Human Factors

Keywords

Agent Software Engineering, Methodologies, Tool Support

1. INTRODUCTION

The Prometheus Design Tool (PDT) [4] supports the design and development of multi-agent systems using the Prometheus methodology [5]. While the methodology supports all phases of development PDT in its current state only supports the system specification and design phases.

The Component Agent Framework for domain-Experts (CAFnE) [1] toolkit was built to allow domain experts with limited programming skills to easily build and modify agent systems. This is a result of the experience gained in working with a group of meteorologists on an agent-based weather alerting system [3]. It is often the case that domain experts are the ones who identify modifications in complex domain applications. However, they have less support for making these changes.

The CAFnE extension to PDT supports the design and implementation of BDI style agent applications. It uses a set of components that define BDI style agents such as event, goal, plan and belief. By adopting a strict structure for these components and visual modelling, CAFnE is able to provide an environment for domain experts to maintain agent systems. The toolkit is based on the concepts of model-driven development and component-based software engineering[2]. Figure 1-(a) shows the layered definition approach used in CAFnE. The toolkit works at the M1 layer and automatically generates the runtime code at the M0 layer.

In addition to using the design artifacts in PDT, CAFnE allows domain experts to use *Attributes* to define the environment and the internal beliefs of each agent of the system; and uses execution units called *Steps* within plans to specify plan behaviour.

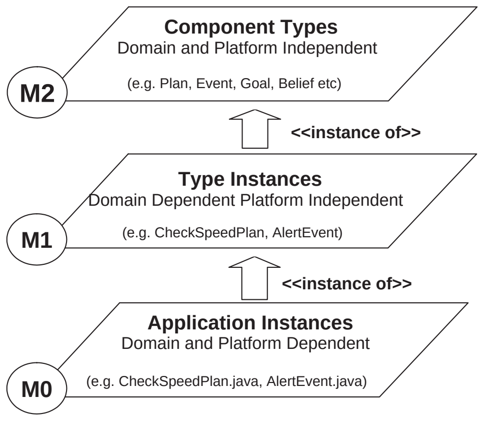
CAFnE takes these components from the design and first creates a domain dependent but platform independent component model of the application. This model is then transformed by a *transformation module* into a specific agent platform that can be compiled and executed. Currently we have developed a transformation module for the JACK[1] agent platform. Similar modules for other platforms can be integrated into CAFnE. Figure 1-(b) provides an overview of the various modules in CAFnE.

2. FEATURES OF THE TOOL

Integrated Development Environment: CAFnE provides a GUI based integrated development environment (IDE) for visual modelling, code generation, compilation and execution of agent applications. In addition to standard features such as design diagrams, CAFnE provides various features that help a domain expert in maintaining an application. The IDE includes an agent instance diagram where initial agent instances can be defined. Users can go into specific details of an agent instance as providing initial belief data values and start up plans to run when the agent instance is created at runtime.

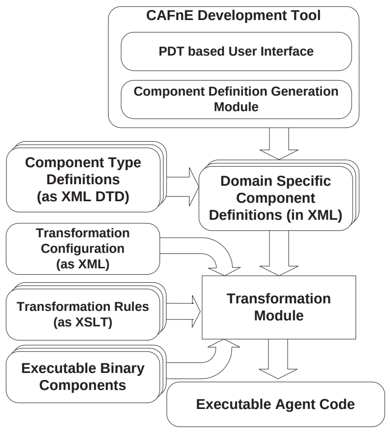
Domain Expert Orientation: CAFnE incorporates various measures in order to allow domain experts with limited agent programming knowledge to build and modify agent applications. CAFnE uses a simplified BDI agent model that is easy to comprehend. By providing diagrammatic representations of the agent application at different levels, CAFnE helps the user in understanding the system. With its automated code generation, CAFnE has reduced programming tasks to editing diagrams and providing necessary parameter values for components. The Attribute based definition of the environment and the internal beliefs, allows the tool to prompt the user with possible parameter or attribute name lists at input boxes. This greatly helps the users in not having to remember parameter names and eliminating errors in input values. The toolkit also has tracing features to help track execution at runtime (see below).

[1]http://www.agent-software.com



Component Types
Domain and Platform Independent
(e.g. Plan, Event, Goal, Belief etc)
M2
<<instance of>>
Type Instances
Domain Dependent Platform Independent
(e.g. CheckSpeedPlan, AlertEvent)
M1
<<instance of>>
Application Instances
Domain and Platform Dependent
(e.g. CheckSpeedPlan.java, AlertEvent.java)
M0

(a) Layered Components Definitions



CAFnE Development Tool
PDT based User Interface
Component Definition Generation Module
Component Type Definitions (as XML DTD)
Domain Specific Component Definitions (in XML)
Transformation Configuration (as XML)
Transformation Rules (as XSLT)
Transformation Module
Executable Binary Components
Executable Agent Code

(b) Overview of the Toolkit Modules

Figure 1: An Overview of CAFnE

Agent Platform Independent: In line with Model Driven Development principles [2], CAFnE uses a platform-independent model (see Figure 1-a). Platform specific code is generated using transformation rules (Figure 1-b). One consequence of this approach is that what the model which the user works with is not specific to a given platform. Changing CAFnE to target an alternative platform is simply a matter of providing alternative transformation rules. This is easiest if the alternative platform is also BDI-like.

Error Checking: CAFnE includes a consistency checker that validates the application model against a set of rules that define the components and relationships between them. In other words, these rules check whether the model created by the user is internally consistent with the meta model used by CAFnE. The errors detected range from warnings such as an agent not having incoming data, to more serious errors such as missing parameters in steps.

Tracing: Basic "Tracing" options provided in the tool help track the execution of the agent system. With event-plan based execution of BDI agents having a lot of asynchronous processes, tracing such events as plan firing, goal raising and belief updating, helps identify errors in the application.

Support for Component Reuse: CAFnE allows for easy reuse of components such as plans, events and beliefs used in the application. Users can copy and paste components across agents and within agents, with the toolkit checking for any inconsistencies in these actions. To support this process the toolkit has features such as automated linking of components when copies are made. Step components used in Plans go beyond the boundary of a single application and allow reuse across applications. It is easy to define additional Steps, and a Step defined in one application can be reused in another application as long as it is provided with the correct input and output parameters.

3. DISCUSSION

We recently conducted an evaluation of CAFnE using five senior forecasters from the Bureau of Meteorology in Melbourne, with varying levels of programming experience, none of whom had developed an agent system before. They were able to rapidly become familiar with the CAFnE concepts and make changes to an existing design for a weather alert system, such as adding the ability to alert on wind data, as well as temperature.

A number of other agent-oriented software engineering methodologies have tool support including Tropos[2] and MaSE[3]. Tropos tool support consists of a number of separate tools that cover different aspects of the software engineering process. None of these tools provide support for transforming design into executable agent code. The JACK Design Environment (JDE) allows design diagrams to be specified and supports automatic code generation and execution. The JDE however is tailored specifically for JACK and requires the user to understand JACK-specific concepts, as well as the JACK programming language.

Currently CAFnE is a stand-alone application developed with the same graphical model as PDT and takes a PDT design file as input. There are still some features that we would like to add to CAFnE, for example, backward propagation (that is, changes in the CAFnE descriptor propagated to the design diagrams) and visualization tools for users to understand the dynamics of a multi-agent system.

This work was done in collaboration with the Australian Bureau of Meteorology and Agent Oriented Software Pty. Ltd. and was also supported by the Australian Research Council (Linkage Grants LP0347025 and LP0453486).

4. REFERENCES

[1] G. Jayatilleke, L. Padgham, and M. Winikoff. Component agent framework for non-experts (CAFnE) toolkit. In R. Unland, M. Calisti, and M. Klusch, editors, *Software Agent-Based Applications and Prototypes,* pages 169–196. Birkhaeuser Publishing Company, Sept. 2005.

[2] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained, The Model Driven Architecture: Practice and Promise*. Addison-Wesley Publishing Company, 2003.

[3] I. Mathieson, S. Dance, L. Padgham, M. Gorman, and M. Winikoff. An open meteorological alerting system: Issues and solutions. In *Proceedings of the 27th Australasian Computer Science Conference (to appear)*, Dunedin, New Zealand, Jan. 2004.

[4] L. Padgham, J. Thangarajah, and M. Winikoff. Tool support for agent development using the Prometheus methodology. In *First international workshop on Integration of Software Engineering and Agent Technology (ISEAT 2005)*, Melbourne, Australia, September 2005.

[5] L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A practical guide*. Wiley Series in Agent Technology. John Wiley and Sons, 2004.

[2]http://www.troposproject.org
[3]http://macr.cis.ksu.edu/projects/mase.htm