# A Model Driven Development Toolkit for Domain Experts to Modify Agent Based Systems[⋆]

Gaya Buddhinath Jayatilleke, Lin Padgham, and Michael Winikoff

School of Computer Science and Information Technology,
RMIT University,
GPO Box 2476V, Melbourne, VIC 3001, Australia
{gjayatil, linpa, winikoff}@cs.rmit.edu.au

**Abstract.** An agent oriented approach is well suited for complex application domains, and often when such applications are used by domain experts they identify modifications to be made to these applications. However, domain experts are usually limited in agent programming knowledge, and are not able to make these changes themselves. The aim of this work is to provide support so that domain experts are able to make modifications to agent systems. In this paper we report on an evaluation of our Component Agent Framework for domain Experts (CAFnE) framework and toolkit, giving a detailed account of a usability study we conducted with a group of experienced meteorologists.

## 1 Introduction

The agent oriented paradigm is becoming increasingly popular for building systems which are relatively complex, and which operate in dynamic domains. One advantage of agent based architectures is that it is relatively easy to extend and expand an application as new conditions are discovered or prioritised. Often there are many nuances in the application domain which are understood by domain experts, but may not be fully captured initially in a requirements analysis. Our aim in the work reported here has been to empower domain experts who take delivery of an agent based software application, to be able to modify and evolve it without the assistance of agent programmers.

To facilitate this we have developed a detailed model of agent based systems that facilitates modelling of the system at a level of detail sufficient to produce code for real applications. Our vision is that a software developer would use this approach and the associated toolkit to develop agent applications. Domain experts who are not programmers (and certainly not programmers of agent applications) would then be able to modify and evolve the application to deal with both growing requirements, and developing understanding of nuances of desired behaviour.

In order to evaluate our approach we have taken a simplified version of an actual agent application developed in collaboration with an industry partner, and implemented it in our system. We have then identified some changes that the actual application had undergone, and have asked domain experts (meteorologists) to attempt to make these

[⋆] 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, LP0453486).

changes using our system. We have observed and recorded these attempts and analysed the extent to which our approach and toolkit appear to be successful.

In this paper we first provide a brief overview of the experimental application, which was a meteorological alerting system developed as part of a collaborative grant with the Victorian branch of the Australian Bureau of Meteorology, and Agent Oriented Software Group. We then briefly describe our approach and toolkit. Additional publications [4, 9] provide greater detail on both the toolkit and the application. The major part of the paper, and its main contribution, is a description of the evaluation of our system based on sessions with five meteorologists. We analyse the success of our approach at four different conceptual levels and conclude that while the user interface could be improved, the approach appears to be quite successful. The fact that the study is based on a real system, and the changes parallel actual changes made to the initial system, lends credibility to the study.

## 2 Overview of the Sample Application

The application on which this study was based is an alerting system which has been developed between 2002 and 2005, as part of a collaboration between the Australian Bureau of Meteorology, RMIT, and Agent-Oriented Software Group. This system, and some of the success in using the agent paradigm has been reported previously [9]. The purpose of the system is to monitor a wide range of meteorological data, alerting personnel to anomalous situations, interactions between data from different sources that may not otherwise be noticed, extreme or escalating situations, and so on. The initial prototype version of the system monitored for discrepancies between data from forecasts for airport areas (Terminal Area Forecasts: TAFs) and data from automated weather stations (AWSs) on the ground at airports. Significant discrepancies resulted in an alert to a relevant human operator.

We reimplemented a simplified version of this system using our toolkit, where data (TAFs and AWSs) were generated by a simulator, and alerts were simply pop-up windows on the machine running the system. The initial system consisted of five agents: one for receiving TAF data, one for receiving AWS data, one for doing discrepancy calculations, and two for providing alerts to end users (one for Melbourne, one for Sydney). We then identified some early changes (or types of changes) that had been made to the actual system, on the request of meteorologists (i.e. domain experts) involved in the project. These included adding an agent for receiving alerts at a new location, alerting on more of the available data, adding the ability to process completely new meteorological data (volcanic ash readings), and adding a more flexible alerting threshold. This then provided the basis for our evaluation activity with meteorologists.

## 3 CAFnE Framework and Toolkit

The CAFnE[1] toolkit supports the generation of complete executable code from a structured model of the application. It is envisaged that an application is developed, by an application developer, using the toolkit to define the relevant conceptual components.

[1] CAFnE stands for Component Agent Framework for domain Experts

It is hoped that due to the intuitive nature of the agent model a domain expert will be able to readily understand the application design, and will in fact be able to modify and further develop it. Because fully executable code is generated based on the model, the domain expert is thus able to modify and extend the application.

### 3.1 Conceptual structure

Starting with the modelling of agents done in SMART [8], and reviewing this against application needs based on our experiences, we developed a simple agent model shown in Figure 1. This model identifies a list of basic component types required for modelling an agent application, namely: *attribute, entity, environment, goal, event, trigger, plan, step, belief* and *agent*. Further details of these can be found in [3].

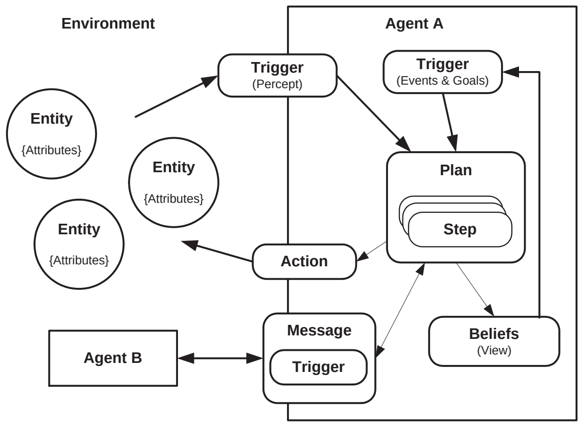


Fig. 1: Simple agent model used in CAFnE

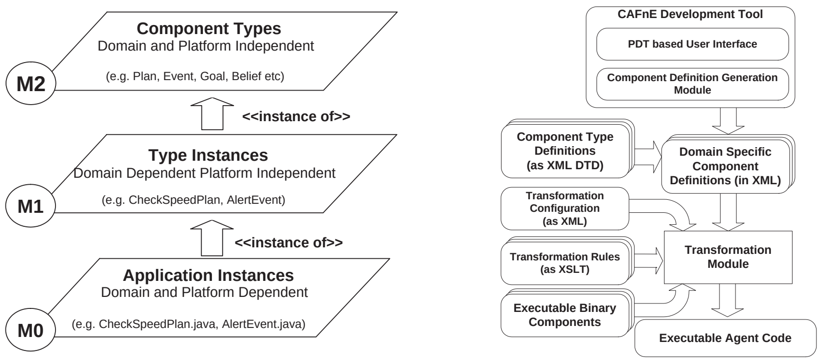
In order to generate executable code from these basic components we adopted a Model Driven Development approach as used in the Model Driven Architecture (MDA) [5] of the Object Management Group (OMG). We use three (M0, M1 and M2) of the four levels used in MDA for application modelling. Figure 2-(a) shows these modelling levels and examples of entities in each layer from the meteorology application.

Each level in the model hierarchy is an instance of the level above. At the meta-meta level (M2 equivalent) we define the domain and platform independent generic component types listed earlier. These generic types are then used in the meta level (M1 equivalent) to define domain dependent component types. This specifies the types of entities required for the particular application domain. The M0 level defines the runtime components of the system which are bound to the domain and also to a runtime platform. In other words, M0 represents the runtime system in a given agent programming language.

We use XML Schema for representing M2, XML for M1 and JACK [1] agent language as M0, the runtime platform. The transformation from M1 (XML) to M0 (JACK code) is done using a set of transformation rules written in XSLT[2].

[2] http://www.w3.org/TR/xslt

Figure 2-(b) gives an overview of the main modules of the toolkit. The Component Definition Generation (CDG) Module is responsible for generating the appropriate XML specifications for the components defined by the user via the UI Module. The output of the CDG Module is a set of XML files that comply with the XML Schema definitions of the component types.



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
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

(a) Layered Components Definitions (b) Overview of the Toolkit Modules

Fig. 2: An overview of CAFnE concepts

The Transformation Module transforms the platform independent XML specifications to executable code in an agent language. This is achieved by applying a set of XML-Transformations (XSLT) to the XML component specifications generated by the CDG Module. Specifics of the Transformation modules including the rules and how it operates are described in [3]. Currently the Transformation Module generates JACK agent language code. However if one wishes to run a CAFnE application in a different agent platform (such as Jadex) it is only required to change the XSLT rules. Thus a technical advantage of the CAFnE platform is that it allows an application to be transformed and run in different runtime platforms without changing the high level application model.

## 3.2 Relationship to Prometheus methodology for software design

Many of the domain independent concepts exist in a range of agent design and development methodologies, which can therefore be adapted for building the application using the CAFnE toolkit. We build on the Prometheus methodology [11] and the support tool available for development using this methodology [10] called PDT [3].

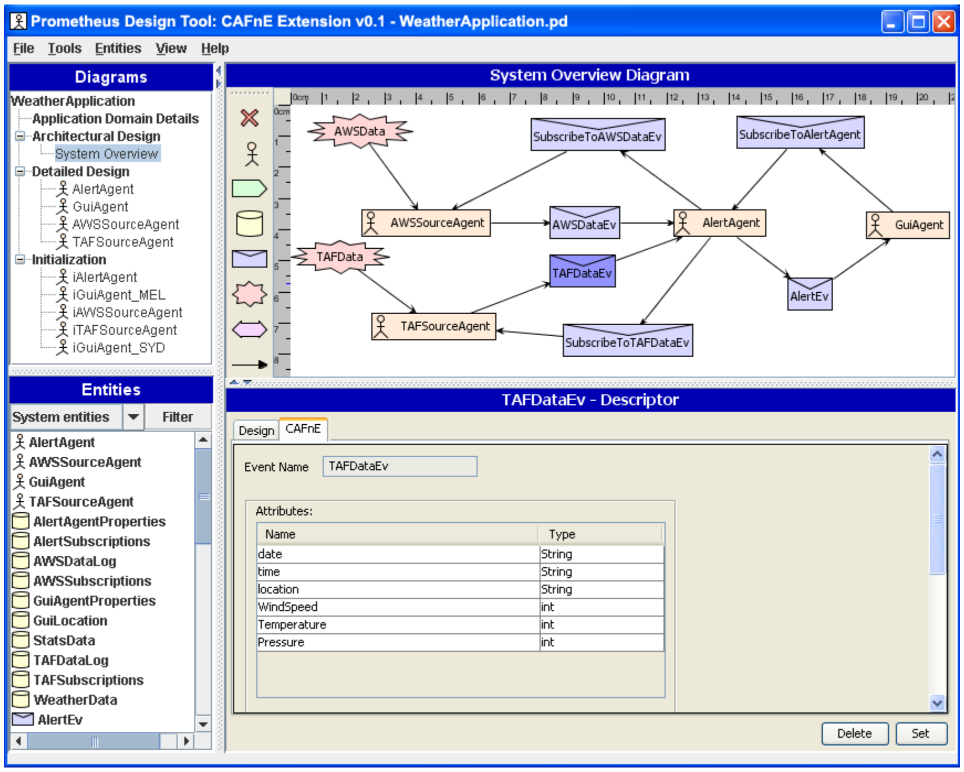
Prometheus supports development of the level M1 entities for *goal*, *event*, *trigger*, *plan*, *belief* and *agent*. In addition, the developer using CAFnE must define the *environment* and its *attributes* as well as the plan *steps*. Steps are executable units used in plans, making it easier to formulate plans. CAFnE constrains (and guides) the developer in modeling the application with these components, thus making it easier for domain experts to understand and make modifications. However CAFnE also provides additional flexibility, by allowing plan steps to have arbitrary target platform code (currently JACK/Java). This provides a mechanism for greater flexibility where needed.

[3] Prometheus Design Tool. (http://www.cs.rmit.edu.au/agents/pdt)

### 3.3 Usage for modifications

Once an application is developed, what the domain expert is provided with is a set of graphical and textual models that present the information from level M1. One of the most important graphical models for an overview of the system is the Prometheus system overview diagram. Figure 3 shows this, (upper right frame) within the toolkit, for the experimental meteorology application.

Clicking on a particular agent type opens up an "agent overview diagram" in this frame which shows the domain dependent types of plans, events, triggers and goals within an agent type. The details of plan steps, attributes and beliefs are available in the CAFnE text based frame at the bottom right of figure 3, called the "Descriptor pane".



Event Name: TAFDataEv

Attributes:

| Name | Type |
|---|---|
| date | String |
| time | String |
| location | String |
| WindSpeed | int |
| Temperature | int |
| Pressure | int |

Fig. 3: PDT based User Interface

The specific agent instances, and their corresponding initial beliefs are accessible from the "Initialization" option in the "Diagrams" pane in the upper left frame (figure 3). When expanded these can be viewed graphically in the upper right frame.

The domain expert who wishes to modify an application does this by interacting with the model available via the CAFnE toolkit. For example to add a new type of agent, this can be introduced graphically into the system overview diagram. Expanding it then allows introduction of relevant plans and events which the agent can handle. The tool also supports copying existing entities (together with their included components) and then modifying. This is a particularly useful way for non programmers to envisage and realise system additions. To add a new instance of an existing type requires addition into the initialization model. Further detail on using the CAFnE tool is available in [3].

## 4 Evaluation Methodology

### 4.1 Participants

In order to evaluate our toolkit and approach, we identified, through our relationship with the Bureau of Meteorology, five experienced forecasters who were willing to spend a couple of hours in an individual interview, using the toolkit. None of the forecasters had been previously involved with our project with the Bureau of Meteorology.

All five participants had at least fifteen years of experience in weather forecasting. Three participants had varying levels of experience in programming (shown in table 1) with only two of them currently being involved in programming activities (shown in the "Comments" column). None of the participants had designed or implemented agent based systems prior to the study.

| Participant | Familiarity | Experience | Comments |
|---|---|---|---|
| A | C, Python | 10yrs | Current work involves programming |
| B | Java | 10yrs | Current work involves programming |
| C | - | 0yrs | No Programming Experience |
| D | C, C++ | - | Past programming experience |
| E | - | 0yrs | No Programming Experience |

Table 1: Domain Expert Profiles

### 4.2 Materials

Participants were emailed a description of the evaluation process along with three documents that provided an overview of the toolkit and the sample application, one week prior to the evaluation session. The documents included a Brief User Guide to the CAFnE Toolkit, a document describing the functions available as plan steps for the sample application and a design overview of the Sample Weather Alerting System. Participants were also given a web link[4] for downloading and experimenting with the toolkit prior to the exercise.

As none of the participants had actually been able to find time to read the documentation or experiment with the toolkit prior to the evaluation interview, the first 30-45 minutes of the interview was spent going through these materials. The participants were then presented with the descriptions of the requested changes along with a description of the observable outcome once each change was successfully made.

### 4.3 Modifications specified

We developed descriptions of four[5] different modifications to be made to the system, which paralleled enhancements or modifications that had been made by programmers to the actual system. The changes were: (1) Show alerts for Darwin - a new city; (2) Add ability to alert on wind data - a new weather data type; (3) Add ability to show volcanic

[4] http://www.cs.rmit.edu.au/~gjayatil/cafne

[5] We actually had five changes, but as no-one had time to look at the fifth change we have excluded it from the study.

ash alerts - a new type of alert and data source; and (4) Change the threshold for alerting from a fixed to a variable value. Of these the third change is the most difficult, and the first is the most straightforward.

Table 2 shows the description of change number 2 and its expected outcome as it was provided to the forecasters.

| |
|---|
| **CHANGE 2:** Add the ability to alert on WindData<br><br>**Background:** The current system only supports generating alerts based on the temperature (data type TEMP) and pressure (data type PRESS) data discrepancies between AWS and TAF. However the AWS and TAF also contain details about Wind (data type WIND) such as wind speed. This data is not used to generate alerts.<br><br>**Required Change:** You are to make the necessary changes to the system in order to generate alerts on wind forecasts. The tolerance level for issuing an alert for a discrepancy between TAF and AWS reading for wind should be 10units.<br>**Expected Outcome:** Alerts on Wind will be displayed |

Table 2: Change Description

### 4.4 Data collection

There were three types of data collected: recording of all verbal comments and interaction; noting of timing information; and a questionnaire[6] which was filled in at the end of each session asking a set of questions regarding how easy they found the tool and the concepts with respect to understanding and realizing the changes. This questionnaire also included a set of questions on the background of the participants regarding their experience in the weather domain, programming in general and agent programming.

While making the changes participants were asked to think out loud and ask questions as they worked on understanding and making the changes. The interviewer (who was the first author) was limited to observing, capturing data and providing assistance on clarifying agent concepts and toolkit functions. The interviewer was not involved in helping the user in any way with deriving design solutions for the changes given. Questions answered by the interviewer included ones such as “How do I copy a plan?”, “Do I have to fill all these description boxes?”, and “How do I change this link from Plan A to Plan B?”. Questions not answered by the interviewer included ones such as “Am I supposed to create a Plan here?”, “Ah! I need an agent instance here, don’t I?”, and “Do I need to send this data to Agent B?”.

Timing information was recorded for each change, starting when a subject began reading the description of the required change and ending when the subject declared it as complete.

## 5 Analysis

Within the available time most participants were able to complete (or attempt) three changes. Table 3 shows the time (in minutes) taken by each subject, and whether the change was successfully accomplished. A ✓ next to the time indicates a successful implementation and ✧ indicates a partial completion.

[6] Available at http://www.cs.rmit.edu.au/~gjayatil/cafne

These results clearly indicate that a domain expert with only a short introduction (of around 40min) to the concepts and the toolkit, is able to make moderately complex modifications to an existing agent system, without the help of an expert agent developer. However some difficulties were also experienced and not all attempts were successful within the time available. We analysed our data in some depth to determine where improvements may be needed in order to facilitate greater ease of use and success.

| Participant | Change 1 | Change 2 | Change 3 | Change 4 |
|---|---|---|---|---|
| A | 15 ✓ | 30 ✓ | 25 ✧ | dna |
| B | 15 ✓ | 30 ✓ | dna | dna |
| C | 20 ✓ | 30 ✓ | 30 ✧ | dna |
| D | 15 ✓ | 35 ✓ | 30 ✓ | 30 ✓ |
| E | 20 ✓ | 25 ✓ | dna | 35 ✧ |

Table 3: Times taken (in minutes)by each subject
(Note: dna = did not attempt; ✓ = complete implementation; ✧ = partial implementation)

In analysing and evaluating the data collected from our interviews, we refer to the cognitive model of program understanding developed by Letovsky et al. [6, 7]. In [6] Letovsky identifies three components in the cognitive model of program understanding. A *knowledge base* that includes programming expertise and problem-domain knowledge; a *mental model* that specifies the understanding of the existing program and an *assimilation process* used to construct the mental model using the knowledge base and stimulus material (such as program code, design documents etc). The *assimilation process* used by programmers in software maintenance is further evaluated in [7] where a Systematic Strategy of understanding a program is claimed to be superior to an As-Needed Strategy in maintenance tasks. The Systematic Strategy refers to a programmer understanding the global view of the application before attempting a modification whereas the As-Needed Strategy refers to limiting this knowledge to the part or parts of the system affected by the change.

In the following we evaluate to what extent the participants were able to develop an adequate knowledge base, in terms of the required agent concepts. We then explore to what extent they have succeeded in obtaining a mental model of the application program, with a focus on a global understanding (facilitating the Systematic Strategy) rather than understanding only of relevant parts (for an As-Needed Strategy). We then look at the participants' ability to make the changes at two levels: the design level covering what changes need to be made, in what components, and the implementation level covering actually realising these changes through the CAFnE toolkit user interface.

## 5.1 Agent Concept Knowledge

The Knowledge Base as identified in the Letovsky model is the understanding of the programming environment and the problem domain. As we are working with domain experts, we assume understanding of the problem domain. Therefore in our case the Knowledge Base evaluation concerns understanding of the agent concepts used in the CAFnE framework.

None of the participants had prior experience in building or modifying agent systems. Subjects spent an average of 40 minutes in browsing the CAFnE user guide and the sample application to acquire this knowledge. Subjects thought out loud while going through the application and seeing the concepts in use appeared to solidify their understanding. When asked if they understood the functions of the basic components (e.g. plan, event, belief) and the agent model (figure1), they commented that the concepts are easy to grasp and intuitive.

However, while the users understood the simple agent model, some of the users faced difficulties and had to clarify while attempting the changes. The majority of the difficulties could be categorised as conceptual problems and interface problems. A common conceptual problem was the difference between the design diagrams (in M1 level) and the instance diagram (in M0 level). Subjects such as C and E, who are new to programming, needed further explanations in how these two levels differ. Another example is the use of Attributes. Subjects needed help in understanding the use of Attributes in Steps and Events, and how they hold values at runtime. While part of this may be attributable to the newness of the ideas, the representation of the concepts in the tool was sometimes confusing and led to interface based problems. For example the Trigger in a plan descriptor form is represented with a drop down list of all incoming triggers. This confused the users as a plan by definition can only have one trigger. In these situations the subjects asked questions to clarify the view they had in mind. These are areas we plan to improve on, especially to narrow the gap between the agent model and the representation in the toolkit.

## 5.2 Program Knowledge

Program Knowledge is the mental model of Letovsky [6]. It refers to the understanding of the existing design of the application. In particular we are interested in whether the participants were able to develop a holistic overall understanding suitable for a Systematic Strategy of modification.

All participants were able to easily understand the high level view (i.e. the inter-agent level) of the application. They were all able to easily explain how the data flowed through the system. CAFnE helps this process by visualising the application at inter-agent (System Overview Diagram) and intra-agent (Agent Overview Diagram) levels. All the participants, first got a broader view of the application using the System Overview Diagram and then drilled down to further details using the Agent Overview Diagram. By following the data flow in these diagrams, subjects were able to develop understanding of the causal relationships between components in the system at different levels such as between agents, between plans and between events. As indicated in [7], the ability to understand the causal relationships between application components forms *strong* mental models that lead to correct modifications. However developing this understanding did require the participants to manually follow through processes based on data flow. One improvement could be to provide some inbuilt mechanism for visualizing the data flow.

Some of the more complex aspects of the design did need clarification. An example is the ’subscription mechanism’ used between the agents to subscribe to services. When explained, they were able to readily understand this architecture and later use it correctly in making the new regional agent instance (i.e. change 1) receive alerts from the AlerterAgent. Another area where users struggled was in understanding what each plan does within an agent. They often referred to plan steps or followed the flow of information in and out of the plan to understand its use. Again, some mechanism that more readily allows visualisation of the dynamic process where plans are used is likely to assist in this. Other aspects which caused some difficulties included the use of start up plans in agents, beliefsets used to hold configuration details and some particular plan steps. It is expected that better design documentation would address many of these issues. Nearly all the users commented that they needed more time with the application, in their words "play around", to understand it better. However, they did feel that it was sufficiently clear and intuitive that, given more time, they would develop a strong understanding.

### 5.3 Conceptual Design of Change

*Conceptual Design of Change* is the derivation of a solution to the problem given in the change description at the conceptual level without actually implementing it using the tool. It is a product of the agent concept knowledge, domain knowledge, program knowledge and the users' capacity to use these in formulating a design change.

In all the changes attempted by users, they were able to come up with a design change relatively easily and rapidly. This was indicated by the thinking out loud practice they adhered to through out the session. Following are some of the common statements made by the users in this stage:

"I need an agent instance for Darwin, don't I?" (attempting change 1)

"I need to make a copy of this plan and this event" (attempting change 2 and pointing to CheckTemperatureDiscrepancy plan and TemperatureData event)

"I have to make an agent type similar to AWSSourceAgent and rename it." (attempting change 3)

The basic agent model used in CAFnE and the Prometheus based graphical notation used to represent the concepts appeared to be successful in providing the understanding necessary to conceptualise the changes required. This is further highlighted by the fact that the partially completed changes shown in table 3 (with a $\diamondsuit$), were all solutions which were correct at the conceptual level, though the users were unable to implement them (within the time frame).

As expected users looked for patterns similar or close to the one they needed in the change. By examining these available patterns users derived partial or complete solutions to the changes. A good example of this is change 2 where users recognised the similarity between Wind data and Temperature data and developed their solution for handling Wind data by looking at how the Temperature data is handled.

While users understood what needed to be done conceptually, realisation of this via the tool interface was somewhat more problematic.

### 5.4 Implementation

Implementation refers to the encoding of the desired modifications or additions using the CAFnE toolkit interface. During this phase users were given assistance when specific questions were asked about carrying out a certain operation, such as "how do I copy a plan?", or "how do I change this link from plan A to plan B?". The assumption here was that more time to find the information in the manual, or greater familiarity, would allow the users to resolve these questions without assistance. However in the context of limited evaluation time it made sense to directly provide this information. However no assistance was given in deciding *what* to do - only in the details of *how* to accomplish it via the tool interface.

Most users indicated that they found the diagrams and the graphical notation easy to understand. They commented that these diagrams reflected how they pictured an agent system, especially the system overview diagram. This was consistent with feedback from a preliminary evaluation with students where the tool did not provide overview diagrams. This lack was a significant issue in gaining an understanding of how individual entities (plans, events, etc.) fitted together and what role they played in the system.

Users heavily utilized the copy and paste functions in replicating patterns similar to the one they needed to implement. Examples include copying and renaming an agent instance in change 1 and copying plans and events in changes 2 and 3. Another useful feature was the ability to transform the application model to executable code and run it, with a click of a button to see the outcome of a change. Two of the users realized after making change 2 that alerts for wind were not being displayed at runtime. With further investigation they were able to find the problem and correct it. Users also found the warnings and errors shown at the transformation level useful. This allowed them to eliminate model based errors such as missing inputs in Steps, and plans without triggers.

The main complaint from the users while using the prototype tool was the lack of features in the user interface, which are normally found in other Windows applications. These included features such as an Edit menu with Undo, Copy, Cut and Paste, right click popup menu for diagram components and drag & drop functions where applicable.

Users like A and B with a programming background had comments on things such as “use of Java standards for GUIs” and “more textual access to steps than with GUI widgets”. Others highlighted the need for using less computer science terms such as “Initialization”, and “Instance” and more training to overcome some of the usability issues. However these do not impact our fundamental concerns regarding the adequacy of the framework and of the tool functionality.

Most users commented that working at the more abstract diagram and form level seemed much easier than directly working with textual code. For example, when shown the Java code generated, user E with no programming experience commented:

“Wow! It’s lot easier than typing all that [Java code]. ” or (user C): “that’s neat stuff [code generation], I mean in the end you could have domain experts, people that just know their job but hopeless at coding do just this [clicking diagrams] and do that [filling forms] and run”

## 6 Conclusion and Future Work

This paper described a conceptual framework of domain independent component types and a toolkit which enables applications to be built and modified using these component types in a structured manner. The approach is consistent with Model Driven Development as fully functional executable code is derived from the models. We have provided a detailed analysis of related work in [3] and found the only similar work being [2]. Most existing agent toolkits are made for experienced programmers and do not provide the same level of support for domain experts. We evaluated our toolkit and approach for modification of agent based applications by domain experts in interviews with five meteorologists, using a simplification of an actual agent based system and changes it had undergone. Our findings could be summarised as: 1) Domain experts with varying programming experience, and with no experience with agent design or programming were able to rapidly (35-40 mins) become familiar with the CAFnE concepts and begin comprehending an agent system design. 2) Users were able to go through the system and understand the functionality from the various diagrams provided. 3) Participating domain experts could make moderately complex changes and run the system, without any prior knowledge of the agent approach or programming. 4) Users found it easier to work at the higher level of abstraction given by the tool, and the overview diagrams provided were seen as useful. 5) Additional support for understanding the flow of processes within the system would probably be helpful. 6) Realizing the changes was hampered by various issues in the GUI, such as non-adherence to user interface standards, due to its prototype nature. In future work we plan to improve the user interface and to provide some mechanism for more readily understanding the role of a component and visualizing the data flow in a particular process.

## References

1. P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. JACK Intelligent Agents - Components for Intelligent Agents in Java. Technical report, Agent Oriented Software Pty. Ltd, Melbourne, Australia, 1998. Available from http://www.agent-software.com.
2. H. J. Goradia and J. M. Vidal. Building blocks for agent design. In *Fourth International Workshop on AOSE*, pages 17–30. AAMAS03, July 2003.
3. 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, 09 2005.
4. G. B. Jayatilleke, L. Padgham, and M. Winikoff. A model driven component-based development framework for agents. *International Journal of Computer Systems Science and Engineering*, 20/4, July 2005.
5. A. Kleppe, J. Warmer, and W. Bast. *MDA Explained, The Model Driven Architecture: Practice and Promise*. Addison-Wesley Publishing Company, ISBN: 0-321-19442-X, 2003.
6. S. Letovsky. Cognitive processes in program comprehension. In *Papers presented at the first workshop on Empirical studies of programmers*, pages 58–79, Norwood, NJ, USA, 1986.
7. D. C. Littman, J. Pinto, S. Letovsky, and E. Soloway. Mental models and software maintenance. In *Papers presented at the first workshop on Empirical studies of programmers*, pages 80–98, Norwood, NJ, USA, 1986.
8. M. Luck and M. d'Inverno. A conceptual framework for agent definition and development. In *Proceedings of the fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL)*, pages 155–176, 1998.
9. 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*, Dunedin, New Zealand, Jan. 2004.
10. 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)*, 2005.
11. L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley and Sons, ISBN 0-470-86120-7, 2004.