20th Int. Conference on Software and Systems Engineering and their Applications, 4- 6 December 2007, CNAM, Paris, France.

An Autonomous Assistant for Architecting Software

José L. Fernández-Sánchez and Javier Carracedo-Pais
Madrid Technical University, Universidad Politécnica de Madrid (UPM),
E.T.S. de Ingenieros Industriales
C/Jose Gutierrez Abascal 2, 28006 Madrid, Spain
Tel: +34 913363144 – e-mail: fernandezjl@acm.org ; javcarracedo@terra.es

Abstract: The CASE tools must be more user-oriented, and support creative problem-solving aspects of software engineering as well as rigorous modeling and model checking based on standards such as UML (Unified Modeling Language). Knowledge based systems and particularly intelligent agents provide the technology to implement user-oriented CASE tools. Here we present an intelligent agent implemented as an independent module of the PPOOA_Visio CASE tool. The agent guides the software architect through a well-established architecting process, suggesting him the actions to be performed and the methodology rules that apply to the current architecting problem context. We describe the intelligent agent, its main capabilities, metamodel and components, and how the agent interacts with the software architect during the architecting process.

Key-words: Intelligent agent, CASE tool, software architecture**.**

.

1. INTRODUCTION

Software engineering is an established discipline of engineering where standards, notations, processes and best practices are currently defined.

The software development process consists of a mixture of creative and mechanistic tasks some of them performed by humans and the others well supported by tools.

The creative tasks of the software architecting phase of a real-time system development are mainly those related to: the identification of system responses, the identification and selection of the architecture components, and the solving of the time and concurrency problems frequent in these applications.

Current CASE (Computer Aided Software Engineering) tools can guide software architects in the software architecting process at the supported methodology level, basically they can check consistency of software models and in some cases verify methodology rules, but they are rather inefficient supporting the creative tasks of the software development.

Knowledge based systems and particularly intelligent agents provide the technology to develop user-oriented CASE tools where an intelligent agent implemented as a tool module, guides the software architect through the architecting process, suggesting him the actions to be performed and the methodology rules that apply to the current problem context.

Based on our previous experience of developing the PPOOA (Pipelines of Processes in Object Oriented Architectures) method and tool for architecting real-time systems, we try to demonstrate that constructing an autonomous assistant or computerized tutor for guiding the development of the architecture of a real-time system is feasible.

We begin the paper describing the characteristics of software agents and their role in software engineering practices. Section 3 introduces PPOOA method and tool. Section 4 presents PPOOA_ATA (Architecting Tutor Agent), its capabilities, the agent metamodel, and the main components of the agent and how it interacts with the software

architect. We then close the paper with some conclusions and how we plan to extend the architecting tutor agent and the PPOOA_Visio CASE tool interacting with it.

2 AGENTS AND SOFTWARE ENGINEERING

Authors involved in agent research have offered a variety of definitions for an intelligent agent. It is not the purpose of this section to survey the definitions of intelligent agents but to identify what is the essence of an intelligent agent, which are its main properties and how an intelligent agent can be applied in the software engineering discipline.

2.1 Intelligent agents

The essence of an intelligent agent is formalized defining it as an entity situated within and a part of an environment, which monitors that environment and acts on it over time, in pursuit of its own plan and so as to change what it monitors in the future.

Franklin identifies the properties of an agent [6]:

- Reactive: responds in a timely fashion to changes in the environment.
- Autonomous: exercises control over its own actions.
- Goal-oriented: may act following a plan.
- Temporally continuous: is a continuous running process.
- Communicative: can establish dialogs with other agents including people.
- Adaptive: sensitive to each user´s strengths and weaknesses.
- Mobile: able to transport itself from one machine to another.
- Flexible: actions are not scripted.
- Character: apparent personality and emotional state.

The intelligent agent definition given above satisfies the first four properties. Adding other properties may produce useful types of agents for example, mobile learning agents.

2.2 Agents and software engineering

Effective intelligent agents must deal with the grey areas of the incomplete function for which it is an approximation. Software engineering is one of these grey areas where the adequate output for a given input is context-dependent. That is, there are different solutions for the same software problem (requirements) and the suitability of a solution (design) depends on the context (project).

The knowledge for building software is buried in books and manuals or in the heads of software engineering experts, and how to find and get access to it is a challenging task. Frequently the software engineer is blocked in one step of the development process, having no access to the human expert that can help and suggest what the software engineer needs to know at this particular situation.

The availability of a computerized tutor is not time restricted and can help the software engineer in three ways: by capturing a significant part of the developing process knowledge; by reasoning based on this knowledge and received events; and by automating the application of this knowledge to the software under development. Presenting relevant information and proposing suggestions eases the decisions made by the software engineer.

Figure 1 represents the interaction of the computerized tutor, the software engineer and the CASE tool in the scenario of generic software development process. The computerized tutor asks questions to the software engineer and receives

responses from him. The computerized tutor checks the software models and receives events from the CASE tool, for example a tool event is received by the computerized tutor when a new building element is drag and drop in a diagram. Following the development goals and reacting to events, the computerized tutor sends suggestions to the software engineer. So, the expert is always available when the software engineer needs help.

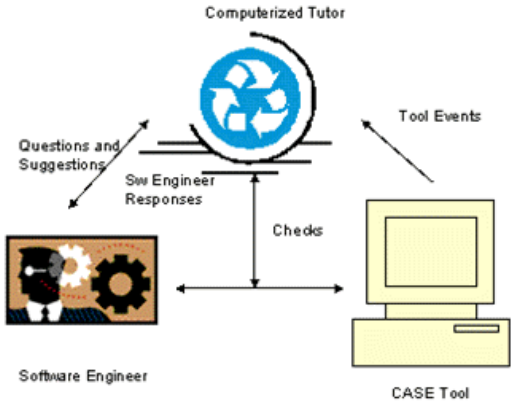

Figure 1. Computerized tutor in software development

The feasibility of the usage of computerized tutors is also supported by recent experiences, which have similarities and differences in goals, scope and implementation with respect our computerized assistant.

Diaz Pace and Campolo report an expert system being constructed that will support design modifiability [2]. Bachmann, Bass, Klein and Shelton propose an expert system that collaborates with the software architect to produce a design of an architecture that supports an expected change [1]. This rule based architecture design assistant uses architecture modifiability attribute models viewed as frames. The next attribute they plan to add will be performance. WayPointer is a commercial tool that provides an integrated solution for implementation of best practices for requirements engineering, modeling, documentation and testing of software. In WayPointer, an intelligent agent is responsible for helping the software engineer to accomplish a well-defined software development task. The knowledge of the agent is captured by a set of rules that are defined in an application specific rule language [10].

3 ARCHITECTING REAL-TIME SYSTEMS

A real-time system (RTS), is one in which correctness depends on meeting time constraints. In these systems, correctness arguments must reason about functional requirements and response time requirements.

Real-time systems often have to deal with multiple independent streams of input events and produce multiple outputs. These events have arrival rates often unpredictable, although they should be processed meeting time constraints specified by requirements.

Object oriented methodologies and early architectural decomposition efforts traditionally focus on domain analysis and functional cohesion to derive software components.

The experiences in defining time constrained architectures, led us to consider that timing concerns must play an important role in the choice of system components and objects. Concurrency modeling and its accompanying coordination (synchronization and/or communication) behaviour become a dominant concern surprisingly early in the architecture development process whenever response time is a critical issue.

In this section, we introduce PPOOA method and tool created late nineties by the first author to solve some problems inherent to RTS architecting [3].

PPOOA uses an extension of UML metamodel and provides the building elements and views for representing a RTS architecture. PPOOA provides the guidelines and architecting process steps for building a RTS architecture, and a later developed CASE tool to create and modify the architecture models and documentation.

3.1 PPOOA architectural style

A software architectural style encapsulates decisions about its building elements and emphasizes important constraints on the elements and their relations. The PPOOA architectural style is described using constructive elements such as components, coordination mechanisms and system responses. Constraints on building elements are represented by guidelines. These guidelines not only represent the semantics of the style, they are also helpful for the architect using the style and are a very important input to build the intelligent agent knowledge.

The guidelines constrain the design space of the software architect. The guidelines are split into the following groups:

- Guidelines concerning the style.
- Guidelines concerning the use of components.
- Guidelines concerning the coordination mechanisms.
- Guidelines concerning system responses.
- Guidelines concerning scheduling policies.

3.2 PPOOA_Visio tool

We implemented PPOOA on the top of Microsoft Visio, considering the extension mechanisms of the tool, and the PPOOA metamodel previously created as an extension of UML metamodel [4].

The UML stereotypes are extended with the elements of the style (periodic and aperiodic processes, controller object, and coordination mechanisms). UML activity diagrams are also adapted for PPOOA requirements, specifically modeling RTS resources and considering scheduling points.

The PPOOA architecture diagram is used instead of the UML component diagram to describe the structural view of the architecture. The architecture diagram focuses on "design or logical components" representation and the composition and usage relationships between them. Coordination mechanisms used as connectors, are also represented [4].

The behaviour view is supported by the "Causal Flow of Activities (CFA)" representation. A CFA represents a system internal view of the flow of activities performed by the system response to an event. The building elements of a CFA are: Triggering event, activity (ies) and continuation elements.

For the behavioural view, PPOOA uses the UML activity diagram with swimlanes to support allocation of activities to the architecture component instances.

Besides the current capabilities of a CASE tool for software development based on UML notation, PPOOA-Visio also supports model checking, documentation generation and an automated help. Additional information about the PPOOA_Visio CASE tool can be found elsewhere [5].

3.3 PPOOA architecting process

To facilitate PPOOA applicability, we created an architecting process, also named PPOOA_AP (PPOOA Architecting Process). PPOOA_AP is focused on the development of the software architecture for a RTS conforming the principles inherent to PPOOA architectural style and using the vocabulary of components and coordination mechanisms proposed by PPOOA. The scope of PPOOA_AP is the architectural or preliminary software development phase in the software life cycle.

PPOOA_AP should be included as a part of the general software development process. As a part of this general process, it takes some inputs from the analysis phase and produces some outputs to the detailed design phase (Figure 2).

The main purpose of the PPOOA architecting process is to produce a rigorous description of the solution that allows the system efficiency evaluation by software architecture assessment techniques, particularly Rate Monotonic Analysis (RMA) [9].

PPOOA_AP is essentially an iterative process that is split into major steps and minor steps. Architecting process major steps are those that follow the identification of the components of the system, their interfaces and the main flows of activities that the system will implement. These major steps represented in Figure 2 include:

1. Identify components and its types.
2. Define interfaces for each component.
3. Describe system responses to events either external, internal or timer.
4. Select the coordination mechanisms needed to solve concurrency issues.

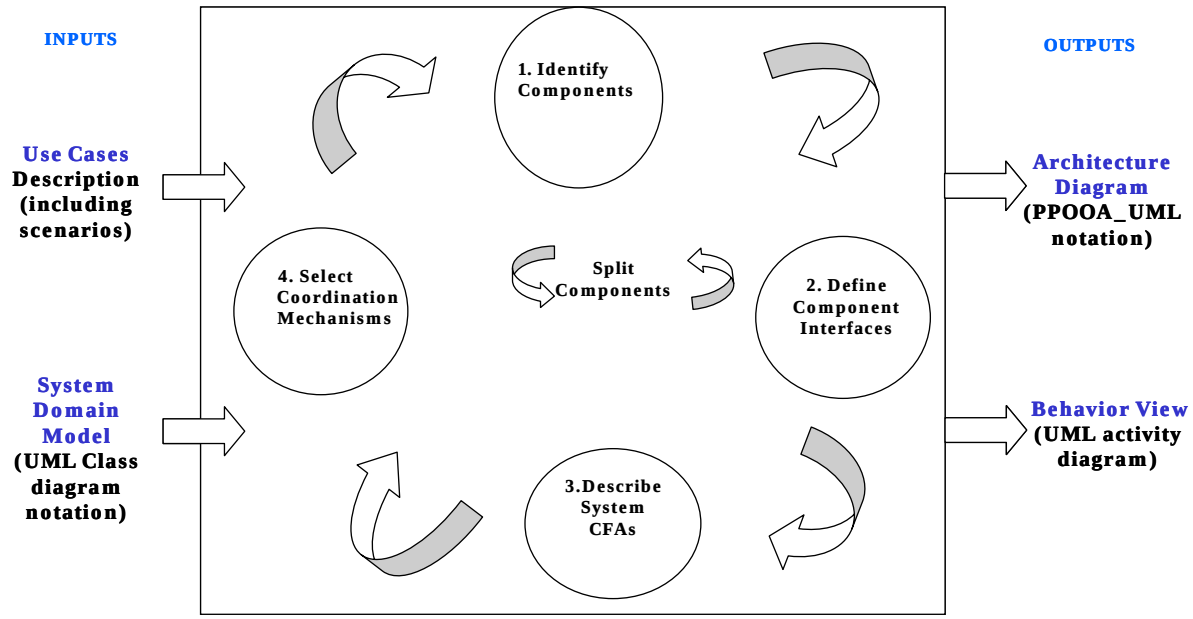

Figure 2. PPOOA architecting process

4 ARCHITECTING TUTOR AGENT

PPOOA_ATA can be considered a hybrid agent exhibiting two different forms of reasoning: one reactive or event driven, and other proactive based on action planning and the PPOOA_AP architecting process execution for achieving the software architecting goals.

We describe below PPOOA_ATA, its capabilities, the agent metamodel, the main components of the agent and how the agent interacts with the user.

4.1 PPOOA_ATA capabilities

The current prototype of PPOOA_ATA supports the following capabilities:

- Has knowledge about the PPOOA architecting process and the architecting guidelines and is able to use this knowledge to give suggestions to the software architect.
- Can proactively take action based on the architecting goals and the architecting process implemented as a plan.
- Can reactively take action based on its interoperability with the PPOOA_Visio CASE tool.
- Can inform the software architect about the current step of the running architecting process or subprocess.
- Help issues regarding PPOOA are presented to the software architect depending on the current context.
- Provide configuration parameters to adapt the agent to the preferences of the software architect.
- The context information of the current software development project is maintained independent of other software development projects supported by the same CASE tool.

4.2 PPOOA_ATA metamodel

The agent metamodel describes the concepts underpinning the agent by means of a UML class diagram. The metamodel presented here was the previous step of the construction of the architecture of the PPOOA_ATA agent. The metamodel represents the agent concepts and their relationships avoiding ambiguities relative to similar concepts.

An important agent metamodel proposed by Kinny, Georgeff and Rao, is BDI [8]. BDI describes the Beliefs, Desires and Intentions held by the agent. Beliefs are the agent´s information about its environment. Desires represent heterogeneous objectives to be accomplished. Intentions represent committed desires. Beliefs can be mapped to a knowledge repository; desires to the agent goals; and intentions can be mapped to plans implemented as actions to achieve the current subgoal [7].

The PPOOA_ATA metamodel is represented in Figure 3; it uses the BDI paradigm and extends it with new concepts that are missing in BDI for example inputs, events and outputs.

The concepts and relationships represented in the metamodel class diagram of Figure 3 are:

- Plan: The agent plan addresses the issue of how the architecting goals are to be achieved.
- Plan Body: The plan has associated a plan body which consists of subgoals and the PPOOA architecting process steps.
- Goal: A goal is a consistent set of desires regarding the architecture models to be produced.
- Action: An action represents an activity that is done. Actions may be the realization of a step of the architecting process or a reflexive action reacting to an event and producing some output to the software architect.
- Output to Architect: The agent as a result of an action can produce either a question or a suggestion or a selection.
- Event: An event is something that occurs in the environment. Here events are either tool events or inputs, for example the creation of a new diagram representing a system response. An event may be an input from the software architect. An event leads to an action.
- PPOOA Help: PPOOA help consists of Guidelines and Help Issues that are applicable to a particular architecting step. A help is associated to an output suggestion to the software architect.

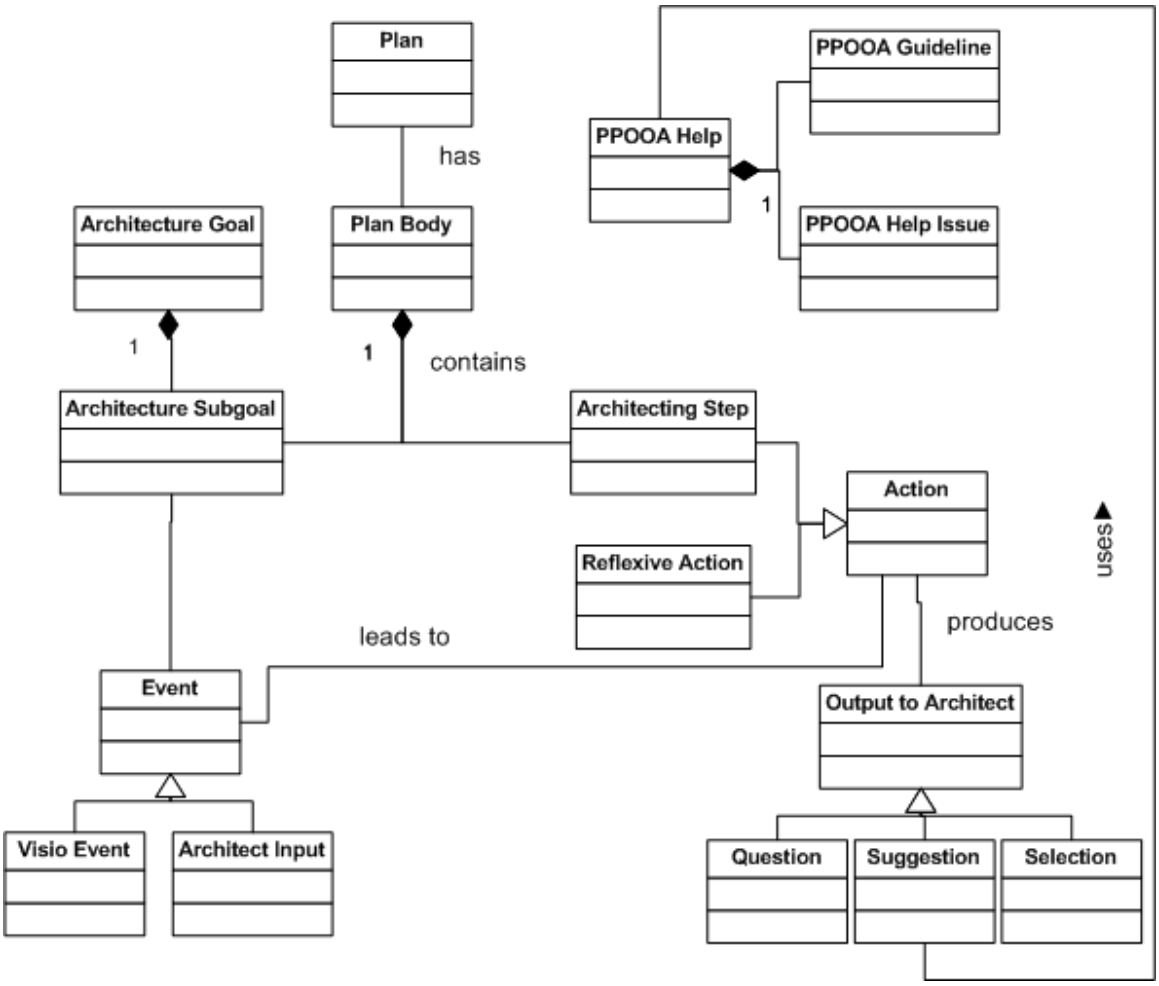

Figure 3. PPOOA_ATA metamodel

4.3 Components of the PPOOA_ATA

The metamodel described above and the integration of the agent with the PPOOA_Visio tool are the main drivers of the architecture solution selected for the PPOOA_ATA intelligent agent.

The agent is integrated in a Windows environment using the Microsoft Foundation Libraries to communicate with the software architect through screen windows and dialogs.

The agent is implemented as an independent add-on of Visio that communicates with the PPOOA_Visio tool using the COM interface provided by the Microsoft Office suite.

The agent is coded in Visual C++ and it executes in its own thread. Thus the PPOOA_ATA agent is autonomous but it can open a PPOOA_Visio instance and communicate with it.

The main components of the PPOOA_ATA agent perform the following responsibilities:

- R1: Managing the sending and reception of messages to PPOOA_Visio tool.
- R2: Analyze each event received from PPOOA_Visio tool.
- R3: Analyze the current step of the architecting process.
- R4: Manage agent configuration changes input by the software architect.
- R5: Guide the software architect.

- R6: Select the PPOOA guidelines applicable to the current context.
- R7: Decide to send an output to the software architect.
- R8: Select and show help issues.
- R9: Store project data (context).

Table 1 represents these responsibilities and how they are allocated to the main components of the agent architecture shown in figure 4.

Agent Controller	R2,R3,R4,R7
Knowledge DB	R6,R8
Interface Visio Tool	R1
Project DB	R9
Configuration DB	R4
Architect Interface	R5,R8

Table 1: Responsibilities allocation.

The main components of the agent are represented in Figure 4 and described here:

- Agent controller managing the current situation based on events and planning next steps.
- Knowledge DB keeping the knowledge regarding the architecting process and guidelines.
- Configuration DB keeping the configuration parameters adapted to the user.
- Project DB keeping the information of a particular software development project.
- Interface Visio Tool receiving events from Visio Tool and communicating with it.
- Architect Interface is the man machine interface.

These components have dependencies among them as represented in figure 4. The agent controller centralizes the control over the rest of the components and interoperates with the PPOOA_Visio CASE tool and the PPOOA help files.

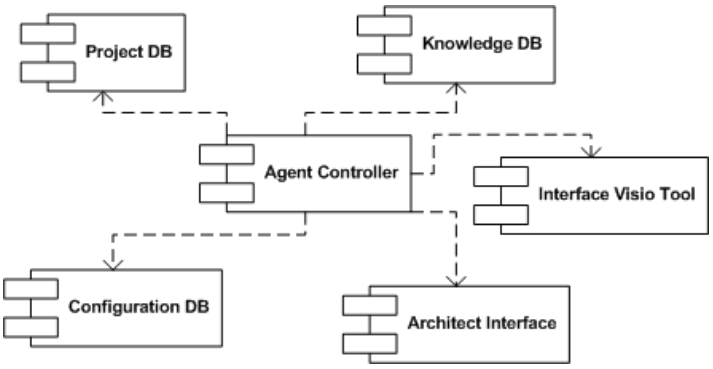

Figure 4. PPOOA_ATA Architecture

4.4 Use of PPOOA_ATA

The PPOOA_ATA agent presents a main screen, shown in Figure5, which is part of the personal assistant and provides input buttons to know the contextual help issues and applicable guidelines. This main screen acts as a real-time library reference tuned to the exact step of the architecting process at which the software architect is working.

The main screen shown in Figure 5 describes the current step and substep of the architecting process; it shows the pending goals and gives the software architect the opportunity to know the help issues (Figure 6) and guidelines applicable to the current context situation.

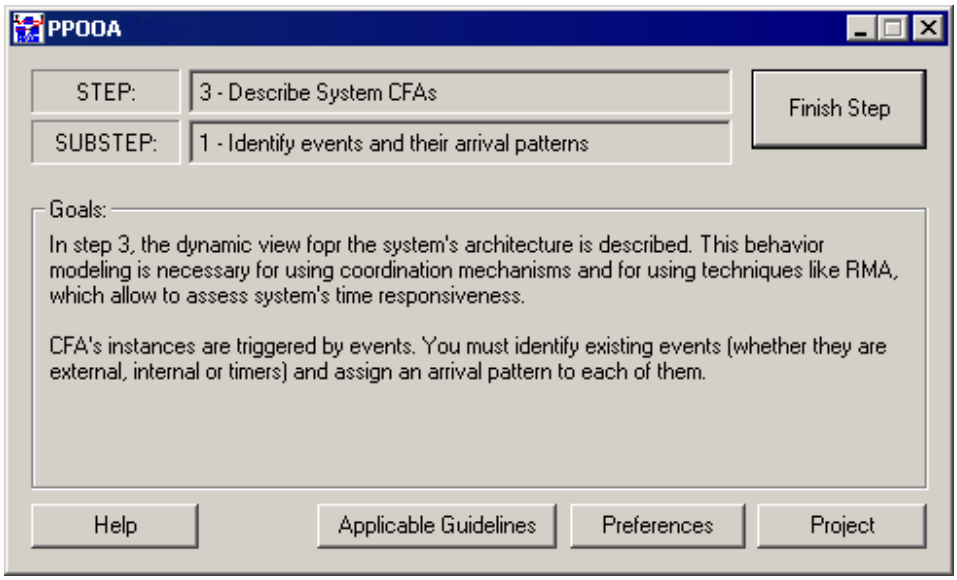

Figure 5. PPOOA_ATA Main screen

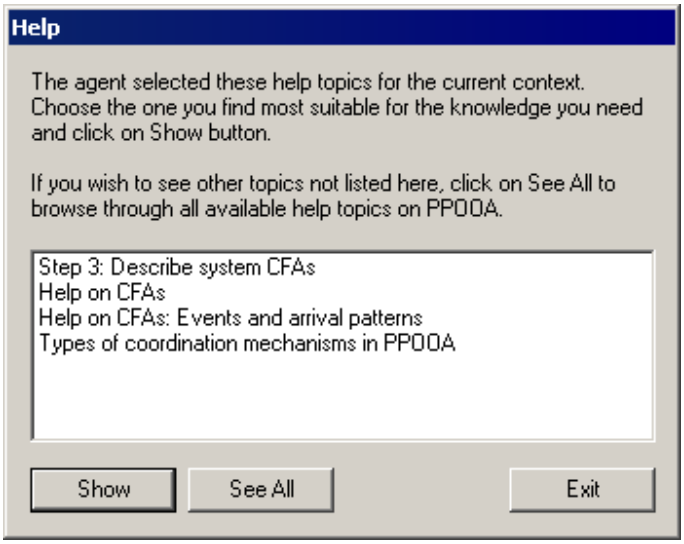

Figure 6. PPOOA_ATA Help Issues screen

The main screen shown in Figure 5 describes the current step, goals and substep of the architecting process for a particular context. In this situation PPOOA_ATA agent recommends the software architect to begin with step 3 of the

PPOOA architecting process. The first substep of the step 3 is concerned with the identification of the RTS external events and their arrival patterns (periodic, bounded, bursty, etc.). The buttons shown in Figure 5 give the software architect the opportunity to know the help issues (Figure 6) and guidelines applicable to this situation.

A complete video demo of PPOOA_ATA can be downloaded from: http://www.ppooa.com.es/ppooa_visio.htm. The demo starts showing the loading of an existing project related to the creation of the software architecture of an elevator control system. The demo shows how PPOOA_ATA loads the context of this software project when it was closed and saved in a previous working session. The context for the demo example is step 2 of the PPOOA architecting process: "Define component interfaces". The different windows related to the current context are showed: contextual help issues and applicable guidelines for example. The interoperability of PPOOA_ATA and the PPOOA_Visio CASE tool can be seen, and the dialog with the software architect is based on questions and output messages that are presented in the demo.

5 CONCLUSIONS AND FUTURE WORK

Our main goal was transform PPOOA_Visio software architecting tool into a more user centered CASE environment. The implementation of an agent acting as an autonomous assistant was one of the main research goals.

We tested the PPOOA_ATA agent with some examples of architecture development performed by university students, and we believe that it performs well regarding tutoring novice software architects based on its interoperability with the PPOOA_Visio CASE tool.

Experimenting PPOOA_ATA with industry projects and obtain feedback is one of the main future issues.

We plan to extend the agent and the CASE tool in the following directions:

- Extend the tool events received by the agent.
- Enhance model checking capabilities of the tool and allow the agent to interpret model checking information and take actions.
- The enhanced CASE tool should offer a wide range of real-time software architecture patterns for reuse.

REFERENCES

[1] Bachmann, F., Bass, L., Klein, M., Shelton, C. Experience Using an Expert System to Assist an Architect in Designing for Modifiability. *Fourth Working IEEE/IFIP Conference on Software Architecture.* IEEE 2004.

[2] Diaz Pace, J.A., Campo, R.M. Design Boots: Towards a Planning-Based Approach for the Exploration of Architecture Design Alternatives. *Argentine Symposium on Software Engineering.* SADIO 2003.

[3] Fernandez, J.L. An Architectural Style for Object-Oriented Real-Time Systems. *Fifth International Conference on Software Reuse.* IEEE 1998.

[4] Fernandez-Sanchez, J.L., Monzon, A. Extending UML for Real-Time Component-Based Architectures. *14th International Conference on Software and Systems Engineering and Their Applications.* CNAM 2001.

[5] Fernandez-Sanchez, J.L., Martínez-Charro, J.C. Implementing a Real-Time Architecting Method in a Commercial CASE Tool. *16th International Conference on Software and Systems Engineering and Their Applications.* CNAM 2003.

[6] Franklin, S., Graesser, A.,. Is it an Agent or just a Program? A Taxonomy for Autonomous Agents. *Third International Workshop on Agent Theories, Architectures and Languages*. Springer Verlag 1996.

[7] Henderson-Sellers, B., Quynh-Nhu, N.T., Debenham, J. An Etymological and Metamodel-Based Evaluation of the Terms Goals and Tasks in Agent-Oriented Methodologies. *Journal of Object Technology, vol 4 no 2, March-April 2005, pp 131-150.* JOT 2005.

[8] Kinny, D., Georgeff, M., Rao, A. A Methodology and Modelling Techniques for Systems of BDI Agents. *7th European Workshop on Modelling Autonomous Agents in a Multiagent World.* 1996.

[9] Klein, M., Ralya, T., Polak, B., Obenza, R., Gonzalez-Harbour, M. A Practitioner´s Handbook for Real-Time Analysis: Guide to Rate monotonic Analysis for Real-Time Systems. Kluwer Academic Publishers, Norwell, MA 1993.

[10] Racko, R. A Cool Tool Tool. *Software Development Magazine. May 2004.* CMP Media LLC.