A Software Engineering Process for BDI Agents

Aaron Hector, Frans Henskens, and Michael Hannaford

University of Newcastle,
University Drive, Callaghan NSW 2308, Australia
{Aaron.Hector, Frans.Henskens,
Michael.Hannaford}@newcastle.edu.au

Abstract. Software agents provide an increasingly popular approach to software engineering, offering a fundamentally different design technique for software development based around the creation and deployment of autonomous software components. One of the leading models for the development of agent-based software is the Belief-Desire-Intention (BDI) model. In order to effectively develop BDI systems, a software development process is required, similar to those that exist for conventional object-oriented software development. This paper presents NUMAP, a development process for BDI multi-agent systems that covers the entire software development lifecycle, from requirements analysis to implementation and testing.

Keywords: Agent-Oriented Software Engineering, Design Processes, Documentation.

1 Introduction

During the last decade there has been an increasing focus on research into software agents. This field, which encompasses a broad range of approaches, has potential for developing applications related to massively distributed systems such as the Internet [1].

Agent-based software engineering shows great promise for software development due to its high degree of abstraction. It has been argued that software engineering supports increasingly complex systems via increasingly sophisticated techniques for abstraction, and that agents may provide the next advance in this regard [2].

The Belief-Desire-Intention (BDI) model [3] is a popular paradigm for development of agents. It assigns "mental attitudes" to agents. The agent's behaviour can be specified in terms of these attitudes, providing an intuitive abstraction for the development of agent-based software.

This paper presents NUMAP (Newcastle University Multi-Agent Process), a comprehensive software engineering process for development of software agents, and in particular, BDI agents.

2 Agent Development Processes

One of the primary concerns of software engineering is developing and formalising processes to aid in constructing complex software systems. As such, agent-based

N.T. Nguyen et al. (Eds.): KES-AMSTA 2008, LNAI 4953, pp. 132–141, 2008.

software engineering requires a process to guide development and formalise best-practices, just as does any other form of Software Engineering [4]. A number of processes and methodologies have been created to achieve this goal [2, 6, 7]. Some of the more popular processes are described in the following sections.

2.1 Tropos

The Tropos methodology [5] is based on two key ideas. Firstly, the concept of agent is used throughout the development process, from early analysis to implementation. Secondly, unlike many agent development methodologies, Tropos deals with the early requirements analysis process, with the goal of attaining a better understanding of how the environment, agents and users interact. It aims to have the entire development process take place at the knowledge level, rather than introducing notions from traditional software development.

There are five main stages to the Tropos methodology: early requirements, late requirements, architectural design, detailed design, and implementation. Early requirements analysis involves: determining the stakeholders within the system and identifying their intentions; determining the goals of each actor; and determining the dependencies between actors.

The "Late requirements analysis" phase defines the system that is to be developed, and specifies how it will interact with its environment. The system is modelled as an actor, and dependencies with other actors in the organisation are defined [6]. The architectural design phase focuses on the system's global architecture, defining the subsystems and their dependencies.

Detailed design in Tropos involves the specification of agents' "micro-level". Agent properties are defined, including goals, beliefs, capabilities and communication with other agents.

2.2 Gaia

The Gaia methodology [2] defines both the macro-level (societal) and micro-level (individual agent) aspects of agent-based systems. Gaia supports the definition of roles, and their assignment to agents in the system.

Gaia has two distinct phases, analysis and design. Analysis is primarily concerned with high-level understanding of the system to be developed. The system's organisational structure is defined through roles and relationships.

Design in Gaia involves the creation of three models for the system. An *agent model*, which defines the agent types and their instances; the services provided by each role are modelled in a *service model*, and the communication between agents is defined in an *acquaintance model*. Design takes place at an abstract level, and is not tied to any specific agent model or implementation environment.

An extension to Gaia that takes an organisational approach to agent design has also been proposed [7].

2.3 Prometheus

The Prometheus methodology [4] attempts to cover the full life-cycle of agent development, from specification and definition of the system to implementation and testing.

Prometheus is divided into three main stages: *System Specification, Architectural Design, and Detailed Design.* Components of the specification and design, such as forms and diagrams, are termed artefacts. At each stage, several artefacts are defined, representing the aspects of the system to be developed at the stage, and are stored as structured data.

The artefacts defined at the system specification level include high-level concepts such as scenarios, system goals, functionality descriptors and basic actions, in addition to the system's inputs and outputs. At the architectural design stage, artefacts include agent types, along with an overview of the system's structure and a description of interactions between agents. At the detailed design level, agents are defined in more detail, along with their capabilities, plans, events they handle, and data stored by the agents.

A support tool has also been developed for Prometheus, to assist developers in following the process.

2.4 Agent OPEN

A different approach to methodology design for multi-agent systems is taken by the OPEN Process Framework [8]. OPEN defines a metamodel for creating customised methodologies, based on the individual needs of a project or organisation. OPEN is based upon the principles of situational method engineering, allowing individual, tailored methodologies to be created.

OPEN provides a repository of predefined method fragments and construction guidelines, which can be used to create personalised methodologies as instances of OPEN's methodology metamodel. The construction guidelines assist the user in selecting the method fragments to be used in the methodology.

OPEN defines various types of method fragment. The five main kinds are: Work Units, Producers, Work Products, Languages and Stages. These are described in [8]. These work units are used to proceed through the activities in a customised process, as developers progress through the life-cycle of the process.

While OPEN was originally designed for object-oriented development, it has been expanded to support agent-based software. This extension is referred to as Agent OPEN [9]. In order to extend the OPEN Process Framework to support agent-based development, a number of additional work units were identified by examining existing agent methodologies, such as Tropos, Gaia and Prometheus. A number of new agent-based method fragments were added to the OPEN repository as part of this examination [8].

3 The NUMAP Process

3.1 Aims

NUMAP is a practical design process that guides the development of agent-based systems. It covers all aspects of design, from early requirements through to implementation. NUMAP provides a set of guidelines that define the basic concepts used in each phase of the design.

In designing NUMAP, a number of desirable properties for agent-based design processes were identified. The first such property is that the process utilises existing tools and techniques, in order to take advantage of prior experience with such techniques. One important existing technique that can be used is goal-based requirements analysis [10], an approach which specifies requirements in terms of goals. Due to the proactive, goal-focused nature of multi-agent systems, goal-based requirements analysis techniques are a natural fit.

Another useful property of a design process is its ability to use a number of different implementation environments. There are a variety of different environments available for implementing multi-agent systems, such as JACK [11], Jadex [12] and Swarm [13], and a design process is evidently more useful if it can be used with a number of such environments. However, care must be taken to ensure that the concepts used in design are not generalised too much, in order to ensure a smooth transition between design and implementation.

A support tool is needed to assist with the process. This tool must allow developers to enter data as they progress through the process, print documentation based on this data, and generate code to be used as the basis for implementation.

The ability to tailor the process is also required. In particular, designers should have the ability to select different requirements techniques, low-level design approaches, and varying code generation tools needs to be supported.

3.2 Overview

Rather than taking an abstract approach to defining the design concepts, NUMAP ensures that they have parallels in real-world agent implementation environments. In doing so, NUMAP allows software engineers to produce a design specification that is closer to the actual implementation, and which takes into account the specific requirements of the agent design technique that is being used.

In order to retain flexibility and support for different agent types, NUMAP uses a modular approach, where particular phases of the process can be replaced with a different module in order to support different design approaches.

For example, the current Agent Design and Implementation modules being used with NUMAP are based upon the BDI agent philosophy. The concepts defined during design are closely related to those used in BDI-based [13] implementation environments such as JACK [11], and Jadex [12]. These could, for example, be swapped for Swarm based [13] agent design and implementation modules in order to support this different agent design approach. The NUMAP process has been carefully designed to allow for such module changes without affecting the rest of the process.

Additionally, the design itself is modular. For example, if a decision is made to change agent implementation environments after the design is complete, then only those parts of the system related to the agent implementation module need be changed. The rest of the design remains unchanged.

The key concepts that are defined in the process are goals and agent types. Agent types define the kinds of agents that can be instantiated, in the same way as classes define the objects that can be created in object-oriented software engineering. Goals define the objectives of the agent type.

NUMAP concepts are described by completing a series of forms within a design tool. Each concept has its own distinct form, which lists the attributes that need to be defined to fully describe that concept.

The process itself is divided into five distinct phases: Requirements Elicitation, Analysis, Organizational Design, Agent Design, and Implementation.

Each of these phases can be altered or replaced in order to support a variety of design approaches. In particular, the Requirements phase can be altered to support different requirements elicitation methods, and the Agent Design and Implementation phases can be replaced in order to support different agent design techniques.

The remainder of this section provides an outline of the NUMAP process and each of its phases.

3.3 Process Outline

The primary activity within each phase of NUMAP is specification of the concepts required for the system model. These concepts are defined by filling in the required information for each concept, usually by entering the information into the support tool. Each phase of the process has its own distinct phase model, and the collection of these creates the overall system model. As development progresses, the models from the previous phases are used to assist in constructing the model for the current phase. The NUMAP tool assists where possible in automating the transition between phases. The model for each phase consists of the elements defined (via forms) for that phase, and the relationships between them.

Diagrams can be generated from the system model to provide a visual representation of the model. They can also be used as an alternative mechanism for specifying details of the system, or for making minor alterations to the design.

Other processes, such as Gaia [2] omit specific design details in order to preserve generality. NUMAP uses a modular approach to allow for a more detailed design process, without restricting the entire design process to that approach. This combines the benefits of a more detailed design process with the advantages of generality.

Different modules may be used for different agents within the same system, in order to allow various forms of agents to co-exist within the same system. For example, some agents in a system may be BDI-based, while others may be Swarm [13] based.

NUMAP's support tool provides a form-based GUI for entering the data for each phase of the process. Additionally, it enforces data integrity by checking content as it is provided, assists with transitioning between each of the phases of the process, and guides the developer through each step of the process.

The support tool assists with validating the correctness of the system model; for example, ensuring the pre-conditions and post-conditions for goals exist, ensuring that two goals are not preconditions of each other, and ensuring all goals have plans associated with them.

The tool also assists with generating diagrams from forms. These diagrams help visualise the system model, and can also be edited to directly update data in the system model.

3.4 Process Phases

An overview of each of the phases of the NUMAP process is provided below. The objectives of the phase are described, and the main concepts for each phase are explained.

Requirements

The Requirements phase uses a goal-based requirements analysis method to describe the requirements for the system. The overall goals of the system are defined, along with an overview of how these goals will be achieved.

Any goal-based requirements method may be used at this stage. The requirements method currently being used with NUMAP is GBRAM [10], however this could be replaced with another approach by using a different Requirements module. Depending on the approach used, the specific concepts defined in this phase would differ.

The GBRAM-based requirements stage requires three distinct sets of data to be defined, as specified in the GBRAM process. Firstly, the *agents and stakeholders* within the system need to be defined. Next, the *goals* for the system are defined. The GBRAM requirements elicitation method provides a number of strategies for defining these goals. These strategies are followed to create a detailed model of the high-level goal hierarchy of the system. Lastly, the operationalisations, or *actions*, of each goal are defined. These provide a basic description for how each goal may be achieved.

Analysis

The analysis process is concerned with creating an abstract model of the system based upon the results of the requirements phase.

There are two primary goals of this phase. Firstly, the outputs of the requirements phase are mapped into a standard format. This is necessary due to the modular nature of the requirements process. Secondly, these design elements are expanded with more detail, and a number of additional elements are defined.

The first of these additional elements is the system's *environment*. In order to define this, relevant elements that are external to the system are identified. The *sensors* that are used to sense environmental elements, and the *effectors* which the agent uses to effect change upon the environment, are also defined.

Also defined are agent *services.* These define the functionality that an agent type makes available to its peers. Services may be grouped into *roles*. Roles are cohesive groupings of agent services that can be applied to agent types that share functionality. Each agent type may have one or more roles, defining the services that the agent provides, and also the services that it uses.

Preliminary *organisations*, used for grouping agent types into subsystems, are also defined at this phase.

At the conclusion of this phase, all of the above essential elements are defined, ready for the organisational design phase.

Organisational Design

The next stage of NUMAP is organisational design. This phase is concerned with defining the system at an inter-agent level. Concepts that were defined in the analysis phase are refined, and the interactions between agent types are defined. The internal functionality for each agent is not defined at this stage.

An organisational-level description of *agent types* is created, expanding on the more simple types that were identified during the previous two phases. This forms the final list of agent types that will be used in implementation.

High-level goals for each of these agent types are also defined at this stage. These are used for documentation purposes only, and provide a brief description of the functionality of the agent.

In order for agents to communicate, *message types* need to be defined. To assist with standards compliance, each message type is assigned a performative, based on those defined in the FIPA Communicative Act Library Specification [14].

Agent communication is further structured by *agent interaction protocols*. These define a formal method for the exchange of messages between two parties. The interaction protocols that are defined may be based on existing FIPA interaction protocol standards, for example Request Interaction Protocol [15], Query Interaction Protocol [16], or Contract Net Interaction Protocol [17].

Higher-level agent functionality is defined by refining the *service* and *role* descriptions from the analysis phase. The particular message types and interaction protocols used by each service are defined.

Organisations are also carried over from the analysis phase, and can be used as a basis for defining more complex structures for the system. For example, extensions to the organisational design phase to allow complex behaviour such as rules, norms and agent admission criteria are currently being explored.

Agent Design

The agent design phase involves defining the internal behaviour of each agent. Depending on the technique used for creating agents, different agent design modules may be used. Currently, a BDI module is being used for creating agents.

The primary concept to be modelled within the agent design phase is the *agent type*. Within the agent design phase, this defines the internal aspects of the agent.

All agent types have *goals* that define their proactive behaviour. An agent will generally have several goals that it simultaneously pursues. An agent attempts to satisfy its goals via *plans*. Each plan defines an action, or set of actions an agent may perform in pursuit of a goal or in response to a reaction.

Agent *reactions* are instant reactions to some event. Unlike goals, these do not run constantly, rather they model the reactive behaviour of the agent type. *Events* are triggered by agents in response to some situation that requires immediate attention. They can be generated by a received message, a belief state change, or they can be manually generated by a plan.

Beliefs define what an agent "thinks it knows". They are the agent's information about its environment and about other agents within the system. That is, beliefs are the agent's symbolic representation of its surroundings.

The agent's *environment* defines the elements external to the system with which the agent will interact. This includes environmental elements that will be sensed, as well as elements that will be changed by the agent. *Sensors* and *effectors* are the agent's mechanism for interacting with its environment. Sensors are used to receive inputs from the environment, while *effectors* are used by the agent to effect change upon its environment.

An agent uses *plan selection rules* to select which plan is used to attempt to satisfy a goal. Similarly, when an agent must use a service, it may have to select from a number of agents that provide that service. *Delegation selection rules* are used to make that selection, and are defined in this phase.

Agent *capabilities* can be used to create a grouping of beliefs, goals, reactions and plans that can be used by any agent. Specific agent functionality can be defined in the capability, and inherited by an agent type, thus encouraging software reuse.

Implementation

Implementation focuses on writing code for the finished design, which will be run in an agent runtime environment. There are a number of such environments, including JACK [11], and Jadex [12].

The close mapping between the concepts defined in the design phases and the actual implementation environments allows for agents to be readily implemented from the design specification.

The NUMAP support tool may be used to generate code for a specific agent implementation platform. It can generate the basic code for an agent, by producing a template, and defining the agent's overall structure. The programmer uses this as a basis for implementing the agent.

3.5 Evaluation

In order to provide a qualitative evaluation of NUMAP, a multi-agent Marketplace Simulation project has been created. This system features a number of interacting Customers, Retailers and Supplier agents involved in buying and selling of goods. The system was initially implemented, using Jadex, in an ad-hoc manner, without the assistance of a design process or support tools. Subsequently, a system with the same requirements was implemented with the assistance of NUMAP.

Development with the assistance was found to be much less error-prone, particularly in defining Agent Definition Files. The final design created with the assistance of NUMAP and its support tool was found to have more consistent interaction between agents, with communications being more structured. Code reuse was increased, with an increase in the number of plans shared between agents, and an increase in shared code for communications and service handling. Documentation created with NUMAP was found to be comprehensive, and greatly assisted with debugging. Total development time was also considerably shorter when using the NUMAP process.

A comparison was made with a system developed using Prometheus. This system was developed using the process outlined in [4] and the Prometheus Design Tool (PDT) 2.5f [18]. NUMAP shows a number of advantages in this comparison. The use of an established requirements analysis technique (GBRAM) allows experience with this technique to be utilised during the requirements elicitation phase of development.

The modular nature of NUMAP provides the flexibility of changing agent design approaches late in the development life-cycle, and allows for easy mixing of different styles of agents within the same system. Additionally, NUMAP presently allows for code generation in Jadex and has preliminary support for JACK, in contrast to PDT 2.5f, which only supports JACK.

Further evaluation will be undertaken in future, to evaluate NUMAP against a number of other processes. A larger-scale problem, namely the implementation of an existing distributed e-commerce system will be used for this evaluation and will be the subject of a future report.

4 Conclusion

NUMAP is a new process with accompanying support tools that provides a modular approach to developing agent-based software, from requirements elicitation to actual implementation. The support tool currently has modules for the GBRAM requirements process [10] and the Jadex agent environment [12]. After evaluation of the current modules is complete, support will be added for additional requirements elicitation techniques and agent implementation environments.

A number of extensions to the process are under development, including an extended organisational design phase, based upon recent work in organisational theory for agent systems [19]. This extension handles enforcement of rules and norms within agent organisations and agent admission criteria.

There are plans for the NUMAP support tool to be expanded to provide additional support for multiple iterations through the process. NUMAP's support for defining links between elements in different phases will assist with this expansion.

References

1. Nwana, H.S., Ndumu, D.T., Lee, L.C., Collis, J.C.: ZEUS: A Toolkit for Building Distributed Multi-Agent Systems. Applied Artificial Intelligence 13, 129–186 (1999)
2. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Autonomous Agents and Multi-Agent Systems 3, 285–312 (2000)
3. Rao, A.S., Georgeff, M.P.: BDI-agents: from theory to practice. In: Proceedings of the First Intl. Conference on Multiagent Systems (1995)
4. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley & Sons, Chichester (2004)
5. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F.: Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Sytems 8, 203–236 (2004)
6. Giunchiglia, F., Mylopoulos, J., Perini, A.: The Tropos Software Development Methodology: Processes, Models and Diagrams. In: Giunchiglia, F., Odell, J.J., Weiss, G. (eds.) AOSE 2002. LNCS, vol. 2585, pp. 162–173. Springer, Heidelberg (2003)
7. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing Multiagent Systems: The Gaia Methodology. ACM Transactions on Software Engineering and Methodology 12, 317–370 (2003)

8. Henderson-Sellers, B.: Creating a Comprehensive Agent-Oriented Methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, pp. 368–386. Idea Group Publishing, Hershey, PA (2005)
9. Henderson-Sellers, B., Giorgini, P., Bresciani, P.: Enhancing Agent OPEN with concepts used in the Tropos methodology. In: Omicini, A., Petta, P., Pitt, J. (eds.) ESAW 2003. LNCS (LNAI), vol. 3071, Springer, Heidelberg (2004)
10. Anton, A.I.: Goal-Based Requirements Analysis. In: ICRE 1996, IEEE, Los Alamitos (1996)
11. Howden, N., Ronnquist, R., Hodgson, A., Lucas, A.: JACK Intelligent Agents – Summary of an Agent Infrastructure. In: The 5th International Conference on Autonomous Agents (2001)
12. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: Implementing a BDI-Infrastructure for JADE Agents. EXP - In Search of Innovation 3, 76–85 (2003)
13. Daniels, M.: Integrating Simulation Technologies With Swarm. In: Macal, C.M., Sallach, D. (eds.) Workshop on Agent Simulation: Applications, Models and Tools, Argonne National Laboratory, Argonne, Illinois (2000)
14. FIPA: FIPA Communicative Act Library Specification (2002)
15. FIPA: FIPA Request Interaction Protocol Specification (2002)
16. FIPA: FIPA Query Interaction Protocol Specification (2002)
17. FIPA: FIPA Contract Net Interaction Protocol Specification (2002)
18. RMIT Intelligent Agents Group: Prometheus Design Tool (2007), `http://www.cs.rmit.edu.au/agents/pdt/`
19. Dignum, V., Weigand, H.: Toward an Organization-Oriented Design Methodology for Agent Societies. In: Plekhanova, V. (ed.) Intelligent Agent Software Engineering, pp. 191–212. Idea Group (2003)