The Styx Agent Methodology*

Tracking Number: 104

Keywords
Agent-based software engineering, methodologies for agent-oriented software development.

ABSTRACT
Agent-oriented software engineering is a promising new approach to software engineering that uses the notion of an agent as the primary entity of abstraction. The development of methodologies for agent-oriented software engineering is an area that is currently receiving much attention, there have been several agent-oriented methodologies proposed recently and survey papers are starting to appear. However the authors feel that there is still much work necessary in this area; current methodologies can be improved upon. This paper presents a new methodology, the Styx Agent Methodology, which guides the development of collaborative agent systems from the analysis phase through to system implementation and maintenance. A distinguishing feature of Styx is that it covers a wider range of software development life-cycle activities than do other recently proposed agent-oriented methodologies. The key areas covered by this methodology are the specification of communication concepts, inter-agent communication and each agent's behaviour activation—but it does not address the development of application-specific parts of a system. It will be supported by a software tool which is currently in development.

1. INTRODUCTION
Agent-oriented software engineering (AOSE) is a promising new approach to software engineering that uses the notion of an agent as the primary entity of abstraction [16]. The agent-oriented approach is rapidly emerging as a powerful paradigm for designing and developing complex software systems. AOSE researchers hope that the use of the agent abstraction will provide a significant improvement to current software engineering practice, similar to the improvements gained from structured programming, the object-oriented approach [2] and design patterns [14].

The development of methodologies for AOSE is an area that is currently receiving much attention, there have been several agent-oriented methodologies[1] proposed recently [12, 23, 8, 11] and survey papers are starting to appear [15]. However there is still much scope for work in this area and the authors believe that the agent-oriented methodologies proposed so far can be improved.

The rest of this paper is structured as follows, the new ideas that Styx introduces are discussed in section 2 and the scope of Styx is outlined in section 3. The Styx Agent Methodology itself is presented in section 4 and it is compared to other methodologies in section 5. Future work is discussed in section 6 and conclusions are drawn in section 7.

2. TOWARDS A NOVEL METHODOLOGY
Software development has been identified as a difficult task, software has a high inherent complexity and its abstract, intangible nature adds further difficulties [3]. Over the past two decades research in software engineering has improved the software development process significantly, nevertheless, many software projects are still late or over-budget [21]. A key idea that has emerged from software engineering research is the use of software development methodologies, which are a set of procedures and methods to guide the software development process. The goal of developing and using such methodologies is to change software development from an ad-hoc practice to a well-structured engineering process that produces high-quality software within the constraints of limited resources and adhering to a predicted schedule.

Recent commercial systems [4, 22] have demonstrated that AOSE is potentially a powerful new software engineering paradigm. However these systems have been developed without the support of agent-oriented methodologies; current methodologies are yet to be widely adopted and may not have been sufficiently mature to be useful in the development of these applications. For agent-oriented software engineering to become a widely accepted practice, as many agent researchers predict it will, it is important that mature tools and methodologies are developed.

*The primary author of this paper is a student.

[1] In the interest of brevity the phrase "agent-oriented methodologies", or just the word "methodologies" will be used in the place of "methodologies for agent-oriented software engineering" unless otherwise qualified.

Requirements ⇓ Analysis ⇓ Design ⇓ Implementation ⇓ Maintenance

Table 1: A typical software development life cycle

Several agent-oriented methodologies have been recently described; High Level and Intermediate Models [12], Gaia [23], the ZEUS Methodology [8] and Multiagent Systems Engineering [11]. All of these offer approaches to the analysis and design of agent-oriented software systems. Ideas introduced in these methodologies have been drawn upon in the development of Styx, however there are several new ideas that distinguish Styx from those previously presented in the literature.

Styx covers a wider range of software development life-cycle activities than other methodologies, providing not only analysis and design models but also skeleton source-code for the implementation phase and support for the maintenance phase. It is designed so that a software tool can automate the transformations between analysis-level and design-level models and automatically generate skeleton source-code from the design-level models. This software tool will also informally verify the development process. Domain concepts that will be used in communication between agents are modelled at the analysis level, allowing a more complete model of inter-agent communication at the design level. Styx also utilises the interaction protocols specified by the Foundation for Intelligent Physical Agents (FIPA) [13] in order to support inter-agent communication.

2.1 Covering the Software Life Cycle

It is customary in the software engineering literature to prescribe a number of phases that constitute 'the software development life cycle', an example is given in table 1. The software development life cycle covers the entire life of a software system, starting from gathering the initial requirements for the system, building models of the system, implementing and finally maintaining the system.

Current work on agent-oriented methodologies typically does not cover all phases of the software development life cycle. The first phase, gathering requirements, does not significantly change for agent-oriented software projects. Techniques for requirements gathering already exist, for example formalised specifications, use cases or user stories. These are generally not developed further for the agent-oriented approach. Note however that the requirements are interpreted using agent-oriented concepts in the analysis phase.

The analysis and design of agent-oriented systems is significantly different from analysis and design of other types of software systems. Although tools are often borrowed from object-oriented approaches, there are many differences between agents and objects that must be considered. This phase has received the most attention from agent-oriented methodology researchers and it forms a substantial part of the Styx methodology.

Implementation

The implementation phase has received less attention; it is not supported in the methodologies reviewed in this paper. Since these methodologies are intended to be useful over a wide range of agent types and there are so many different types of agents, it is difficult for a methodology to be both widely applicable and also support the implementation phase.

To address this issue Styx introduces the idea of an abstract agent specification language. This is to be a text based language that is sufficiently generic that it can be mapped to a large proportion of agent-oriented software development frameworks and toolkits. In the ideal case these mappings would be performed automatically, however in the case that an unsupported framework or toolkit is used a mapping could be achieved by hand or a new automatic mapping be developed.

The implementation phase of the software development life-cycle involves transforming the set of design level models into an executable implementation defined in some programming language. Supposing the ideal case, providing automated support for this phase would involve some form of automatic programming. However this is not generally feasible because including enough information in the design level models for a program to be automatically generated would make the design phase far too complex. To ensure that it remains focused on the task at hand, Styx does not attempt to provide support for developing application-specific parts of a system, but rather focuses solely on the agent-oriented aspects. The authors believe that the implementation phase can be best supported by providing a skeleton implementation of the agent-based aspects of a software system, leaving programmers to focus on the application-specific aspects of their system. Note that the generation of this skeleton implementation would be a two-step process; firstly the design models would be mapped into the abstract agent specification language, and then the abstract specification would be mapped to a particular implementation language, toolkit or platform.

Automatically generating such a skeleton implementation directly from the design-level models would require that they contain a sufficient amount of information about how the individual agents are structured and how they will behave. The Internal Agent Model and Conversation Model of the High Level and Intermediate Models methodology and the Services Model of the Gaia methodology all provide a table that loosely resembles a finite state machine. The authors believe that it would be feasible to automatically transform models such as these to some skeleton implementation.

Maintenance

The longest phase of the software development life cycle is usually the maintenance phase. A well-documented development process is important for maintenance. However aside from providing such documentation there has been no support for maintenance in existing methodologies.

Documents outlining the analysis and design phases of system development form an important part of the support offered by Styx for the maintenance phase, as is true of other methodologies. However the skeleton source code generation proposed in the previous paragraph provides an additional area in which the maintenance phase needs to be supported. During a system's lifetime changes may be made to the analysis and design models, as it is likely that requirements will change over time and thus require modifications to these models. Styx provides a source-code skeleton to support the implementation phase, which is generated from these analysis and design models. Now since these models will change over the lifetime of the system it is important that the methodology provides some way of updating the implementation skeleton in response to such changes. Simply regenerating this skeleton would not provide sufficient support for this, as the skeleton will be fleshed out with application specific code during implementation. Thus an important feature of the proposed methodology is to be able not only to generate these skeletons but also to be able to update them with the application specific code in place.

2.2 Designing for Automation

Several parts of the High Level and Intermediate Models methodology appear to be partially automatable, however the authors believe that a methodology designed specifically to be supported by computer software could be more automatable, especially by providing automatic techniques for the transformation between the design and implementation phases. A key feature of Styx is the software tool that supports this methodology.

Note that automatic programming is not the goal of this methodology; application specific aspects will be left to the designers of each system. This tool will provide support for drawing the various graphical models in the methodology, generate and maintain skeleton design models and skeleton implementation source code, and perform informal validation of the development process.

2.3 Roles as Agent Classes

A strength of the object-oriented paradigm is the ability to specify reusable classes of objects, rather than specifying individual objects, which promotes reusability and modularity. The Gaia methodology, as well as other work [17], uses roles in a similar manner for agent-based systems. Developing with roles means that one agent may be able to play several roles, or a single role may be played by more than one agent. Styx is strongly oriented towards developing reusable roles that are later used for the construction of agents.

2.4 Specifying Agent Message Content

It is non-trivial to develop a mapping between the content of agent communication language messages and the objects[2] that the application specific parts of the system will work with [7]. Neither of the methodologies proposed here address the specification of the content of agent message, however this has been included in the ZEUS methodology [8].

Styx incorporates a model at the analysis level which specifies the possible content of agent messages. This model uses the class diagram syntax of the Unified Modelling Language (UML) [20]. UML class diagrams have been chosen because they are a well-known modelling representation, and because of the increasing usage of the object-oriented paradigm in the professional software development community, both for application-specific coding and also for agent development toolkits and frameworks. Modelling message content as object classes makes the conceptual gap between application specific code and the content of agent communication language messages smaller. This idea draws upon recently published work [9, 10].

2.5 Reusing FIPA Interaction Protocols

Current agent-oriented methodologies either do not specify the conversations that will occur between agents in detail (for example Gaia), or otherwise expect that the system designers will invent new conversation protocols for each agent system (for example High Level and Intermediate Models and MaSE [11]). Specifying agent conversations is an important part of a design methodology that seeks to support the implementation phase, however it seems that reinventing conversation protocols for each agent system is often unnecessary. Styx will incorporate the specification of conversation protocols at the design level, but will draw these from a well-known pool—the interaction protocols introduced in the forthcoming FIPA 2000 specifications [13]. These cover a wide range of possible inter-agent interactions, from simple Request and Query protocols, to more complex Dutch Auction and Iterated Contract Net protocols. Although the complete documentation for these is not yet available, nevertheless based on the strength of previous versions of the FIPA interaction protocol specifications and on the wide range of proposed protocols the authors believe that these protocols will form a sound basis upon which to build this part of the methodology. Note that reusing these protocols does not restrict Styx to agent systems that are based on the FIPA specifications because, with some slight modification, the protocols could be used with different standards for message passing that exist in other types of agent system.

3. SCOPE OF THE STYX AGENT METHODOLOGY

Styx is intended to be used in the development of collaborative agent systems [18]. These are systems which tend to use static, coarse-grained agents and are typically used to solve problems more efficiently than a single centralised system. Problems may exceed the capabilities of a single centralised system because of resource limitations, the need to interoperate with multiple legacy systems or because a problem is inherently distributed, for example distributed sensor networks, distributed data sources, or distributed expertise.

Styx does not consider systems that contain a large number of roles—it is envisioned that applying Styx to a system that has significantly more than about twenty to thirty different

[2]Assuming that the system will be implemented in an object-oriented programming language, which is currently a common approach.

roles would result in the analysis level models becoming too complex to give a clear, high-level overview of the system. However there is little restriction on how many agents can be instantiated from the roles defined from Styx, for example a system may contain hundreds of telephone agents instantiated from a single role.

It is assumed that agent systems developed using Styx will be implemented using some agent development toolkit or framework that is based on an object-oriented language[3] and that application specific code will also be developed in the same object-oriented language.

Such notions as planning, scheduling, mobility and learning, which are commonly associated with agent systems, are not explicitly handled by Styx. It is assumed that support for these things would be provided either by the agent development toolkit or framework, or that they would be included by-hand in the application-specific parts of the system.

4. OVERVIEW OF THE STYX AGENT METHODOLOGY

A schematic overview of Styx is presented in figure 1. Styx is briefly summarised in the next paragraph, and more complete descriptions of each part of the methodology are given in sections 4.1 to 4.6 with reference to a simple fruit-market scenario.

The analysis phase starts by identifying agent roles and domain concepts, followed by generating a high-level Use Case Map [5], which gives an overview of the entire system, and a Domain Concepts Model, which specifies what concepts will be used in the communication between agents. In the design phase Role Responsibility Models are generated for each component of the Use Case Map, where each responsibility of a component is specified in more detail. The Role Relationship Model specifies how roles are related to each other and the concepts about which they will communicate. The Deployment Model maps the roles identified in the Use Case Map to agents. The implementation phase is supported by an Agent Skeleton and together with application specific code this forms the final implementation. Although it is not shown in the figure the maintenance phase will be supported by both the models detailed so far and also by changes in the analysis and design models being reflected in the implementation skeletons.

Fruit-Market Scenario

A simple multi-agent fruit-market scenario will be used to demonstrate the proposed methodology. This involves buyers and sellers of fruit in an electronic marketplace. A seller can notify the marketplace that some fruit is for sale with a sale-order and buyers can notify the marketplace they want to buy fruit with a buy-order. The marketplace attempts to match buy- and sell-orders; and when a match is made it notifies the buyer concerned, who then sends payment details to the seller, who in turn sends delivery details for the order to the buyer. The actions of the buyer and seller agents are directed by human users, intended users are farmers who

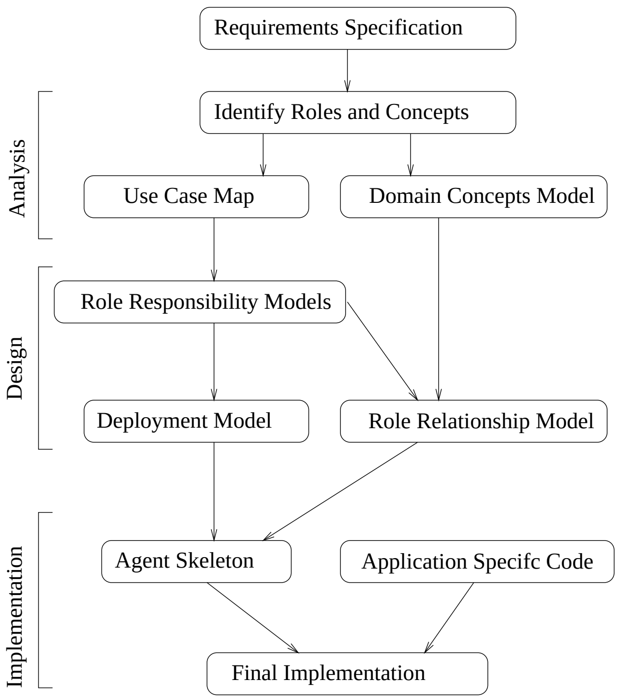

Figure 1: Overview of the Styx Agent Methodology

will sell fruit, supermarkets that will buy fruit and warehouses that will both buy and sell fruit. When placing an order with the market, buyers and sellers name the type of fruit, specify a quality rating (A, B or C grade), the price per unit and the quantity. The marketplace assumes that buy orders can be filled by more than one seller, that sell orders can be split between buyers, and that all transactions are successful.

4.1 Use Case Map

The first step of the analysis phase is to create a high-level Use Case Map (UCM) [5] for the system. Use Case Maps model possible processes in a system as paths which traverse various components of the system. Components are drawn as boxes, while paths are drawn as lines crossing various components (see figure 2). The start of a path is indicated by a solid circle, while the end point is indicated by a strong line. When a path crosses a component that component is assigned one or more responsibilities associated with the path. The Use Case Map is labelled with component names, responsibility names and other explanatory notes. These maps provide a highly condensed notion suitable for modelling the high-level behaviour of a system.

To develop the Use Case Map, the roles that agents may play are identified in the requirements specification and placed in the Use Case Map as components. Interactions between roles are then identified in the requirements specification and placed in the Use Case Map as paths. Finally when a path crosses a component, one or more responsibilities are assigned to that component. Thinking of components as agent roles and paths as interactions between agent roles means that the Use Case Map becomes more of an agent-oriented model, as opposed to the standard object-oriented

[3]Such toolkits are becoming increasingly common, for example JADE [1], FIPA-OS [19] and JACK [6].

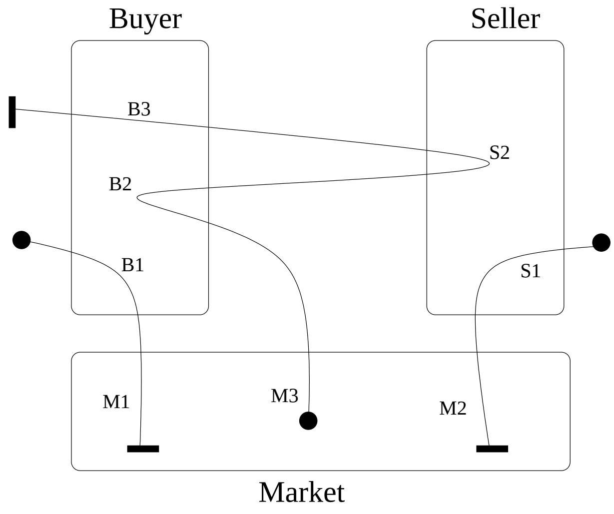

Figure 2: A UCM for the fruit-market scenario.

Label	Responsibility
B1	Post buy-order
B2	Send payment
B3	Receive fruit
S1	Post sell-order
S2	Receive payment and send fruit
M1	Receive buy-order
M2	Receive sell-order
M3	Match buy- and sell-orders and notify buyer

Table 2: Responsibilities for the fruit-market UCM

interpretation outlined in Buhr's original work [5].

A sample UCM is given for the fruit-market example in figure 2 and the responsibilities in this figure are expanded in table 2. This UCM was developed by first creating system components for each identifiable role in the requirements specification and then for each interaction between entities a path was drawn on the diagram and responsibilities were assigned.

4.2 Domain Concepts Model

The Domain Concepts Model models each concept that agents will communicate about. Each concept is modelled as a particular object class, expressed using a UML Class Diagram. A Domain Concepts Model for the fruit-market example is presented in figure 3. This shows object classes for buy- and sell-orders and orders that have been filled.

Note that the types assigned to fields will be later mapped to implementation level types, and that this example uses `String` and `int` for simplicity. More complex field types would ideally be used in a more complete model, for example `deliveryInfo`, which is modelled as a String in this example, would be better modelled as another object class, with address, date and time fields, which themselves may be object types.

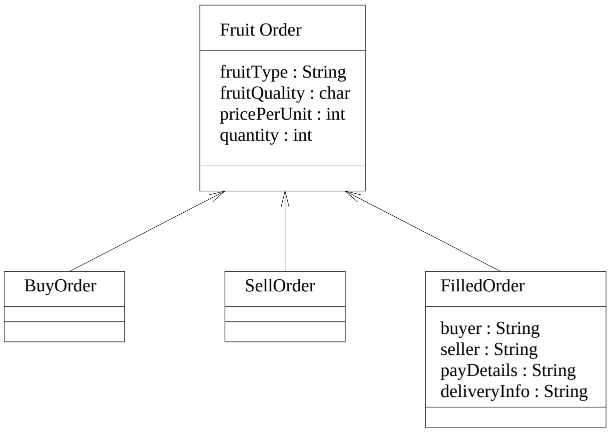

Figure 3: Domain Concepts Model for the fruit-market scenario.

4.3 Role Responsibility Model

Role Responsibility Models are created for each component of the analysis level UCM, and take the form of a table with four columns: responsibility, pre- and post-condition, and action. For each responsibility in the analysis level UCM, an entry is made in the appropriate role's Role Responsibility Model. The pre- and post-conditions are informal statements that can be either true or false. These are listed to guide the implementation phase; the implementation-level action will be performed when the pre-conditions are true, and a correct implementation will achieve the post-conditions unless it meets some error condition. One or more actions, which are simply named at this stage, are listed for each responsibility. These give some indication of the actions that are to be performed in order to achieve this responsibility. Several actions can be specified, allowing a responsibility to be broken down into a number of steps at this level.

There are several keywords used in the Role Responsibility Model; the pre-condition `message received`, post-condition `message sent` and action `sendMessage`. These keywords will be used during the generation of skeleton source code to link the actions specified in the Role Responsibility Model with the agent interactions specified in the Role Relationship Model.

A sample Role Responsibility Model for the fruit-market scenario is shown in table 3. Because of space limitations the responsibilities (abbreviated to Resp.) are named using the abbreviations presented in table 2.

4.4 Role Relationship Model

The Role Relationship Model further elaborates the relationships between roles that are indicated by the analysis level Use Case Map; relationships exist where there is a path linking two components of the Use Case Map. Each relationship is assigned a type drawn from the interaction protocols specified in the forthcoming FIPA 2000 specifications [13] and an object from the Domain Concepts Model. The interpretation is that a conversation of the specified type will occur between the agents, where information is interchanged using the specified object.

Role name: Marketplace

Resp.	Pre-condition	Post-condition	Action
M1	Message received	Order added to database	addOrderToDB
M2	Message received	Order added to database	addOrderToDB
M3	Buy and Sell orders matched	Message sent	sendMessage

Role name: Seller

Resp.	Pre-condition	Post-condition	Action
S1	User wants to sell	Message sent	sendMessage
S2	Message received	Payment checked, Message sent	checkPayment, sendMessage

Role name: Buyer

Resp.	Pre-condition	Post-condition	Action
B1	User wants to buy	Message sent	sendMessage
B2	Message received	Message sent	makePayment, sendMessage
B3	Message received	User knows shipment details	notifyUser

Table 3: Role Responsibility Models for the fruit-market scenario

A sample Role Dependency Model is shown in figure 4 for the fruit market scenario. This shows several 'information' dependencies, where one role makes some information available to another role. For example a Buyer would inform the Marketplace when it wishes to buy fruit, using a Buy-order object. Note that the 'information' dependency is used here as a place holder for the appropriate FIPA interaction protocols.

4.5 Deployment Model

The Deployment Model is the most simple model in this methodology. It specifies a many-to-many mapping between agents and roles which assigns roles to agents. This model specifies what agents will exist in the system, and what roles they will play. An example for the fruit-market is given in figure 5.

4.6 Implementation Phase

The design-level models and the Domain Concepts model will form the basis for generating skeleton source code for the implementation phase. The generation of this skeleton implementation will be a two step process, firstly the design models will be mapped into the abstract agent specification language, and then the abstract specification would be mapped to a particular implementation language, toolkit or platform.

The Domain Concepts Model will map directly into a set of object class definitions that will be available to all agents. Programmers will be able to use these object classes for communication between agents without concern for how

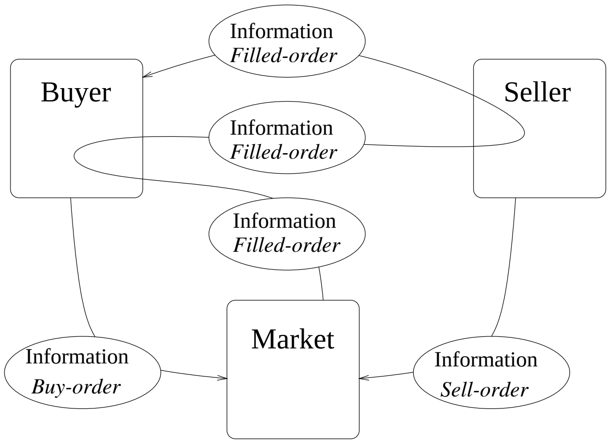

Figure 4: Role Relationship Model for the fruit-market scenario.

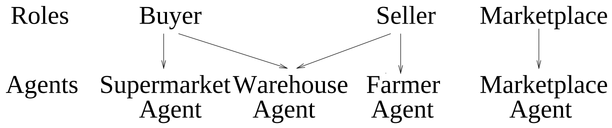

Figure 5: Deployment Model for the fruit-market scenario.

the object instances are mapped into a string based representation for the particular agent communication language encoding.

The Role Responsibility and Role Relationship Models will be used to generate skeleton code that takes care of inter-agent conversations and behaviour activation. The skeleton code will provide method stubs for each action specified in the Role Responsibility Model and these stubs will be filled in by programmers. The Role Relationship Model will be used to generate state-machine-based entities that take care of conducting the conversation appropriately and that activate the appropriate action methods when a certain message is received. When an action method is activated, it will be supplied with the instance of an object from the Domain Concepts Model that arrived with the activating message. Programmers will be provided with methods to call when it is necessary to send a message from within application-specific code, these methods will have a formal parameter of the type specified in the Role Relationship Model (drawn from the classes in the Domain Concepts Model).

So far the source code skeletons for agent roles have been outlined, however these must be mapped to individual agents for the deployment of the system. This is specified at the design level in the Deployment Model, and at the implementation level this information is used to group roles into agents. How this occurs will depend largely upon the target agent development toolkit or framework.

5. STYX AND OTHER METHODOLOGIES

There are several parallels between Styx and other recently proposed agent-oriented methodologies, however most of the ideas that are reused are given a significantly different interpretation and several new ideas are introduced.

Styx draws on three models that exist in the High Level and Intermediate Models methodology, however Styx interprets the components of these models as roles rather than individual agents. The approach of using a Use Case Map at the analysis level was seen as a particularly good way of giving a high-level overview of the system in a single diagram, which details both the structural and the behavioural aspects. In contrast to the High Level and Intermediate Models methodology, Styx interprets the components of the Use Case Map as roles, rather than as individual agents.

The Role Responsibility Model is similar to the Internal Agent Model of the High Level and Intermediate Models methodology and by Gaia's Services Model. Representing the responsibilities of a role as a table is a useful way of specifying the actions that carry out the responsibility at a design level, and since these tables resemble finite state machines they can be used to generate skeleton source-code.

The Role Relationship Model is inspired by the Dependency Diagram of the High Level and Intermediate Models methodology, however new ideas have been introduced; relationships are defined using FIPA interaction protocols in preference to ad-hoc dependency types, and information about discourse objects is included. The Role Relationship Model is somewhat more complex than the original Dependency Diagram, however reusing FIPA interaction protocols and using object classes specified at the analysis level allows a more complete model of agents' social behaviour than the combination of both the Dependency Diagram and the Conversation Model of the High Level and Intermediate Models methodology.

The Deployment Model is drawn directly from the Agent Model in the Gaia methodology. It is necessary to have a model of this type in a role-based methodology as the final products of the methodology are agents, not roles, and Gaia's Agent Model appears a particularly concise way of achieving this.

6. FUTURE WORK

The Styx agent methodology is still in a specifcation phase; many of the difficult problems have been left for future work. A key part of this work will be using Styx to develop examples of a variety of typical agent-based systems, ranging from process control systems, to distributed information systems, to more complex market-place applications than the simplistic fruit-market scenario. This will ensure that Styx is applicable to a wide range of problem domains, rather than being focused on a certain class of applications.

The specification of the text-based abstract agent specification language has not yet been completed. This will be 'boot-strapped' by identifying common concepts identified among current agent software development toolkits and frameworks. This work is perhaps one of the more important areas of Styx, as it will ensure that Styx is not only applicable to current frameworks and toolkits, but also that it will be applicable to future work. JADE [1], FIPA-OS [19] and JACK [6] are the current candidate implementation toolkits to which mappings from the abstract specification language will be provided. The exact detail of how the abstract agent specification will be generated from the design-level models will depend on the specification language that is developed, and is another item for future work.

It is not yet determined if specifying a single object class per relationship in the Role Relationship Model will be appropriate in all cases, for example a 'query' conversation may require one class of object to specify the query but another class for the results of the query to be made available. This problem could be overcome by specifying a single object that has fields for both the query and answer, but this would be more of a work-around than an elegant solution. This issue will be resolved once the FIPA 2000 specifications for interaction protocols have been released, and it may well be necessary to specify multiple object classes per relationship in the Role Relationship Model for certain interaction types.

7. CONCLUSIONS

Styx represents a new approach to agent-oriented software engineering that is more comprehensive in scope than many other methodologies. Styx will provide a software tool that automates much of the difficult work that currently goes into building agent-based systems and will ensure that software development projects using an agent-based approach will be able to focus most of their effort on developing application-specific code, rather than grappling with agent research issues. However there is a significant amount of work, both programming and research, to be completed before Styx is ready for widespread use.

8. REFERENCES

[1] F. Bellifemine, A. Poggi, and G. Rimassa. Jade programmers guide. http://sharon.cselt.it/projects/jade, June 5 2000.

[2] G. Booch. *Object Oriented Analysis and Design with Applications*. Addison Wesley, 1994.

[3] F. P. Brooks. *The mythical man-month : essays on software engineering*. Addison-Wesley, anniversary edition, 1995.

[4] T. Buchheim, G. Hetzel, G. Kindermann, and P. Levi. A multi-agent approach for optical inspection technology. In R. Mizoguchi and J. Slaney, editors, *PRICAI 2000, Topics in Artificial Intelligence*, volume 1886 of *Lecture Notes in Artificial Intelligence*. Springer, 2000.

[5] R. Buhr and R. Casselman. *Use Case maps for Object-oriented Systems*. Prentice Hall, 1996.

[6] P. Busetta, R. Rnnquist, A. Hodgson, and A. Lucas. Jack intelligent agents - components for intelligent agents in Java. In P. Davidsson, editor, *AgentLink News 2*. www.agentlink.org, 2000.

[7] G. Bush, M. Purvis, and S. Cranefield. Experiences in the development of an agent architecture. In C. Zhang and V.-W. Soo, editors, *Design and Application of Intelligent Agents (proceedings of PRIMA 2000)*, volume 1881 of *Lecture Notes in Artificial Intelligence*. Springer, 2000.

[8] J. Collins and D. Ndumu. Zeus methodology documentation. Available at http://www.labs.bt.com/projects/agents/index.htm.

[9] S. Cranefield and M. Purvis. UML as an ontology modelling language. In *Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-23/cranefield-ijcai99-iii.pdf.

[10] S. Cranefield and M. Purvis. Extending agent messaging to enable OO information exchange. In R. Trappl, editor, *Proceedings of the 2nd International Symposium "From Agent Theory to Agent Implementation" (AT2AI-2)*, pages 573–578, Vienna, April 2000. Austrian Society for Cybernetic Studies. Published under the title "Cybernetics and Systems 2000".

[11] S. A. DeLoach and M. Wood. Developing Multiagent Systems with agentTool. In N. Jennings and Y. Lesperance, editors, *Intelligent Agents VI*, volume 1757 of *Lecture Notes in Artificial Intelligence*. Springer, 2000.

[12] M. Elammari and W. Lalonde. An agent-oriented methodology: High-level and intermediate models. Presented at AIOS 99, available at http://www.aois.org/, 1999.

[13] Foundation For Intelligent Physical Agents (FIPA) web site. Located at http://www.fipa.org/.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns : elements of reusable object-oriented software*. Addison-Wesley, 1995.

[15] C. A. Iglesias, M. Garijo, and J. C. Gonzalez. A survey of agent-oriented methodologies. In J. P. Muller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V*, volume 1555 of *Lecture Notes in Artificial Intelligence*, pages 317–330. Springer, 1998.

[16] N. R. Jennings and M. Wooldridge. Agent-oriented software engineering. In J. Bradshaw, editor, *Handbook of Agent Technology*. AAAI/MIT Press, 2000.

[17] E. A. Kendall. Agent roles and role models: New abstractions for multiagent system analysis and design. In *Proceedings of the International Workshop on Intelligent Agents in Information and Process Management*, Germany, September 1998.

[18] H. S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(3):1–40, September 1996.

[19] S. Poslad, P. Buckle, and R. Hadingham. The FIPA-OS agent platform: Open source for open standards. In *Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, pages 355–368, 2000.

[20] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[21] I. Sommerville. *Software Engineering*. Addison-Wesley, 1996.

[22] D. Vasko, F. Maturana, A. Bowles, and S. Vandenberg. Autonomous cooperative factory control. In C. Zhang and V.-W. Soo, editors, *Design and Application of Intelligent Agents (proceedings of PRIMA 2000)*, volume 1881 of *Lecture Notes in Artificial Intelligence*. Springer, 2000.

[23] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3, 2000.