A Methodology for Creating Ontology-Based Multi-Agent Systems with an Experiment in Financial Application Development

Abstract

The creation of an ontology for a particular business domain has advantages for application development, interoperability, reusability, and integration. Agent systems with an underlying ontology can exploit these advantages by providing semantically-aware applications. However, there is a lack of support for ontology development in existing methodologies for designing and building multi-agent systems for business applications. We describe a methodology and an experiment for building such systems. The approach focuses on the development of ontology as the driving force of the development processes and strives to put application development in the hands of domain experts, requiring as little as possible from software developers. We describe two complementary aspects of the approach: (i) a phased, iterative methodology and (ii) a pipeline of select tools with which to carry out the methodology. We report results of an experiment in developing an application for finance, including an evaluation, lessons learned, and recommendations.

1. Introduction

Ontology is a specification of objects, concepts, entities, and entity relations in a particular domain of interest. Alternatively, ontology is defined as a formal, explicit specification of a shared conceptualization [1,2]. It asks the question “What is there?” for a domain, where the answer may involve multiple ways of object composition and varying degrees of object abstraction. Often an ontology is over- or under-specified, and thus pragmatics for a particular application come into play.

Ontologies are used increasingly in software applications for business purposes, particularly in applications that involve methods in artificial intelligence, information retrieval, natural language processing, and knowledge engineering [3]. Further, in the context of multi-agent systems (MASs), the development of an underlying ontology contributes to the analysis and design phase of system development, the intelligence of individual agents, the communication and collaboration among agents at runtime, and the interoperability of agents with other system components such as databases, web-based information, and external applications in an integrated framework.

Traditional software engineering methodologies, however, generally do not support ontology-based development [4,5]. The growing use of ontologies for MASs requires that traditional software engineering methodologies evolve into a set of ontology-driven methodologies. The role of such a methodology is to assist domain experts, software architects, designers, and developers in all phases of the life cycle of ontology-based application development. We describe such a methodology in this paper, called the Methodology for Developing Ontology-Based Multi-Agent Systems (MOMA), and we evaluate MOMA towards application development in the finance domain.

Traditional mathematical methods used to study financial market behavior such as statistical analysis have been identified as having shortcomings such as the following [1]:
1. They describe the macroscopic properties of a system, but the origin of these properties.
2. They fail in situations where the assumptions of mathematical models are not valid.
3. They do not adequately handle the heterogeneity of trade practices.

In dealing with the dynamics of collections of entities, multi-agent systems are better equipped to handle different kinds of global dynamics that can result from these entities impacting each other through their interaction within changing environments.

Of the financial agent systems described in the literature, we found that most of the agents in these models are intrinsically algorithmically linked with mathematical functions dictating and modifying the
agents’ behaviors. That is, financial domain knowledge and business logic is implicit in the algorithms and embedded in the agent code. By placing explicit domain information in the agents, any potential to reuse the multi-agent infrastructure in conjunction with different domains is destroyed. Hence, our research question is whether it is possible, and to what extent to develop an ontology-based multi-agent system such that financial system behavior can be controlled through ontologies defined by financial analysts with little software expertise.

The work of this paper can be classified as design science, one of the two core paradigms that characterise much of the research in the information systems discipline, the other being behavioural science. The design science paradigm seeks to create innovative artifacts through which the development and use of information can be effectively and efficiently accomplished. Artifacts can be broadly classified as methods (i.e. set of steps, guidelines or algorithms), models (i.e. abstractions and representations), constructs (i.e. vocabularies and symbols) and implementation (i.e. prototype systems). Our work aims to create two of these artifacts: models and methods. The models will be the set of ontologies that accompany the MOMA methodology, while the method will be the MOMA process itself.

Typical design science research is comprised of two basic processes: build and evaluate. Build refers to the constructions of artifacts – i.e. the model and the method of MOMA. The evaluation process refers to the use of appropriate evaluation methods to assess the artifact performance. Categories of evaluation methods are Observational, Analytical, Experimental, Testing, and Descriptive. The evaluation method used for our research is Observational through case studies in financial services.

In particular, the design objectives of MOMA are the following:
1. Develop a domain-specific ontology for reuse and sharing among applications in the domain.
2. Move business logic and domain knowledge from underlying agent code to higher level ontology development.
3. Facilitate the use of tools to accelerate ontology-based application development.
4. Distinguish between the domain expert and agent developer, and allow more contribution from the former in application development.
5. Ultimately enable complete application development by domain experts without the agent developer.

The paper is organized as follows. Section 2 provides a review and critique of existing MAS development methodologies that make use of ontology. Section 3 describes MOMA from two complementary perspectives: as an interactive phased methodology and as a pipeline of select tools for application development. Section 4 discusses an experimental application of MOMA towards developing applications in the domain of finance. Section 5 provides an evaluation and lessons learned from our experiments.

2. A review of ontology-based software engineering methodologies

A number of methodologies have been proposed to assist the analysis and design of multi-agent systems. These methodologies vary in their scope, approach, processes, modelling concepts, modelling notations, and intended purposes. Developers of these methodologies generally extend existing methodologies to include the relevant aspects of agents. Examples include Societies in Open and Distributed Agent spaces (SODA) [6] and PROMETHEUS [7]; however, these methodologies do not incorporate ontology concepts [8].

Four MAS development methodologies do indeed incorporate ontology concepts, although not to the full extent as used in MOMA. These methodologies are:

- MAS-CommonKADS [9]
- MESSAGE [10]
- Multi-agent Systems Engineering (MaSE) [11]
- Process for Agent Societies Specification and Implementation (PASSI) [12]

MAS-CommonKADS uses ontologies to represent the knowledge of the application domain and an agent’s local domain-related knowledge. It illustrates the use of ontologies for knowledge representation during agent modelling. However, it does not recognise the role of ontologies in agent communication and it doesn’t include ontology-sharing by agents or the formulation of messages via shared ontological concepts.

Similar to MAS-CommonKADS, MESSAGE uses ontologies as the representation mechanism for modelling an application’s domain knowledge and an agent’s local domain knowledge. MESSAGE makes it possible for agent reasoning to use ontology-based knowledge at run-time. However, it does not recognise the importance of ontologies in agent communication. It doesn’t include an apparatus for ontology-sharing between the agents and external components.

MaSE is an improvement over MESSAGE and MAS-CommonKADS in that it recognises the essential role of ontologies in agent communication. In particular, it requires the developer to formulate the exchanged messages in terms of the concepts obtained from an ontology. MaSE also uses ontologies to
support interoperability. It considers the case of agents committing to heterogeneous ontologies, e.g. agents wrap around heterogeneous information sources, and it highlights the need for ontological mappings between these local ontologies. MaSE’ support for reusability is enhanced since it allows legacy heterogeneous components to be reused. However, the benefits of ontologies for agent reasoning are not realised.

PASSI also uses ontologies to model domain knowledge and an agent’s local knowledge. It supports the use of ontologies for agent communication as well. For each agent conversation, an agent developer identifies the ontology that needs to be shared by the communicating agents and defined messages in terms of shared ontological concepts. However, PASSI doesn’t provide support for the use of ontology-based knowledge by agent reasoning at run-time.

Existing methodologies do not comprehensively implement all of the roles of ontology in MAS development. They don’t use ontology to support interoperability, reusability, agent communication, and agent reasoning in a coherent way.

3. MOMA: A methodology for ontology-based multi-agent application development

MOMA is driven by ontology. It consists of two main development phases: Ontology Development and Agent Development. See Figure 1. During the first phase, domain knowledge is modelled in the form of ontology so that it does not have to be defined in lower level code during agent development. The resulting ontology is then used as a basis for the second phase, which involves the implementation of the agents and the application environment for a specific application.

Figure 2 shows the iterations of domain knowledge as it transitions from ontology to Java code. In this diagram, a pipeline of tools facilitates each of transition. Initially, the domain expert collects multiple sources of knowledge. A tool based on Grounded Theory (GT) assists modellers as they identify the concepts, relationships, and attributes in the domain. For example, “Trader” and “Buy” are concepts which can be implemented as agents and agent actions respectively. After discovering the essential concepts in the domain, the domain expert models the domain’s ontology through the use of the Protégé ontology development tool and translated the ontology into Java code through a Protégé plug-in, Bean Generator. The output is Java code that that is re-used in an ontology-driven application.

Figure 2: Iterations of ontology development

At a lower level of abstraction, the Ontology Development phase is broken down into three steps:
1. **Concept Identification** – Identify the concepts, relationships, and attributes in the domain.
2. **Ontology Modelling** – Model the domain for a specific application.
3. **Code Generation** – Generate code that can be used in the Agent Development phase.

The identification of concepts and relationships for the purpose of ontology modelling is difficult and time-consuming [13]. To make it easier for domain experts who do not have expertise in knowledge engineering, MOMA employs principles in Grounded Theory (GT) [14]. GT is a general methodology that
uses an interpretive approach for deriving theory. It facilitates the production of core categories and relationships from data through a systematic method of continuous comparison and refinement of concepts. Although it originates in the social sciences, it is valuable when applied to ontology construction for specific domains [15]. See Figure 3.

![Figure 3: Methodology of the GT guided tool](image)

Open Coding involves re-factoring, breaking down, examining, comparing, conceptualising, and categorising data to identify a set of discrete concepts which are the basic units of analysis. These codes are usually keywords or essential domain related terms. For example, Stock, Equity, and Shares may be relevant concepts in the domain of financial applications. Once concepts are identified, they are grouped together to establish preliminary categories. Open coding is continuous. As the domain expert gathers new data and identifies concepts and categories, those concepts undergo cycles of comparison to existing concepts in a process called constant comparison.

Axial Coding extends the initial set of concepts and categories by establishing new connections among concepts and sub-concepts. For example, we may have Stock, ASX Code, and Portfolio as initial concepts. These concepts may be further refined as follows: First, we identify the properties. We see that ASX code can be a property of Stock. For identification of relationships, the domain expert will identify the relationship and then connect the two concepts using the relationship. Stock is a part of Portfolio. The domain expert will identify the relationship “a part of” and then link the two concepts. There might well be inverse relationships. i.e. Portfolio “contains” Stock.

Selective coding involves the consideration of multiple concepts and sub-concepts that emerge from axial coding and the identification of core categories to which all child concepts or sub-concepts relate. The core concepts become the means for building a conceptual model from which to develop an ontology.

GT does not specify the form of input data; however, this point is important in ontology development. For example, text from literature and an interviews with a domain expert will require different methods of data extraction. Hence, our approach introduces data collection as a first step before the coding process. Collecting data involves the identification of possible sources of knowledge and converting the data to a common format. The preferred format is text, for which we have built a tool, but other formats in audio or visual are possible.

The domain expert uses Protégé and the Bean Generator to implement an ontology from the GT-guided tool and convert it into Java code for agent implementation [16]. An application executes the code at run-time. The Bean Generator generates Java files that can be used with any FIPA-compliant MAS framework [17]. Figure 6 in the next section shows a graphical representation of the ontology in Protégé as a step in our experiment towards developing applications in the financial domain.

4. An experiment in the financial domain

The growth of e-Finance and web-based financial applications call for the development of intelligent applications with a better understanding of financial semantics, hence the use of MASs is growing. To date, agent-based applications in finance specify domain and agent knowledge through low level programming. In previous sections, we proposed an alternative approach in which domain knowledge is decoupled from agents through the use of ontologies. This approach contributes towards the building of semantically aware intelligent services using ontologies rather than low level programming. Further, it facilitates the possibility of sharing and reuse of the domain knowledge.

Our experiment with MOMA focuses on the creation of applications for e-Finance, in particular for financial market analysis. A similar experiment with MOMA focused on applications for e-Health [8].

Traditionally, market analysis has been a manual process. Several commercial tools are used to facilitate these tasks involving input data in classic spreadsheet style, using relational databases, and statistics toolboxes, but where the bulk of reasoning and intuitive inference is performed by human analysts. Because today most financial information is stored and
accessible electronically, we can build intelligent agents to perform some of the tasks that were previously performed by humans. By defining a financial ontology as a common framework for finance applications, developers can reuse the framework for alternative applications. The purpose of our experiments is to evaluate MOMA towards this end.

Conventionally, research into financial markets involves analytical frameworks with an underlying mathematical theory. However, traditional mathematical methods such as statistical analysis have the following shortcomings [18]:

1. They are able to describe macroscopic properties of a system already in existence, but not the origin of these properties. This type of analysis involves studying financial data from different financial markets and then identifying regular patterns or laws governing the statistics of the data. It usually does not include examining the imperatives and actions that produced the financial data to begin with.

2. They cannot be easily applied to situations where the assumptions behind mathematical equations no longer hold. The majority of statistical methods and techniques developed to analyse data are applicable on the condition that the variables involved satisfy certain assumptions - for example, that the sample comes from a normal population. However, in cases where the assumptions do not hold, these methods cease to be valid and hence should not be used.

3. They do not handle heterogeneity in populations well. Traditionally, the behaviours of traders have been described with mathematical models and their interaction with financial market analysed under equilibrium conditions. In reality, it is not always the case that financial markets and traders exhibit the rational behaviour reflected in mathematical models. Traders display heterogeneity in their trade decision-making, interpretation of company announcements and market trends, and adaptive behaviours.

Thus, alternatives to pure mathematical methods, such as agent-based models or non-parametric methods, are gaining popularity in financial applications [19,20,21]. In dealing with the dynamics of collections of entities, agent-based models are equipped to handle the different kinds of global dynamics that can result from interacting with each other and interacting with a changing environment.

Of the financial agent systems described in the literature, the agents in these models are linked via mathematical functions dictating and modifying the agents’ behaviours [22]. Further, the domain knowledge is embedded in the agents. By placing most of the explicit domain information within agents, any potential to re-use the multi-agent infrastructure for other applications is impractical.

The main objective of MOMA is to separate the domain knowledge from underlying agent code through the use of ontology. This approach allows domain experts and users of the system to dictate the behaviour of the application through high level concepts rather than low level programming languages. Another benefit is the potential for sharing and reuse of the domain knowledge.

The Zeta model calculates the chances of a public company going bankrupt within a two-year time period [23]. The model produces a number called the company's Z-score which is a reasonably accurate predictor of future bankruptcy. The model is:

\[ Z = 1.2A + 1.4B + 3.3C + 0.6D + 1.0E, \]

where:

- \( A = \text{Working Capital/Total Assets} \)
- \( B = \text{Retained Earnings/Total Assets} \)
- \( C = \text{Earnings Before Interest & Tax/Total Assets} \)
- \( D = \text{Market Value of Equity/Total Liabilities} \)
- \( E = \text{Sales/Total Assets} \)

The zeta model returns a Z-score to represent the likelihood of a company going bankrupt in the next two years. The lower the Z-score, the more likely a company is to go bankrupt. A Z-score lower than 1.8 indicates that bankruptcy is likely, while scores greater than 3.0 indicate bankruptcy is unlikely to occur in the next two years. Companies that have a Z-score between 1.8 and 3.0 are in the gray area, i.e. bankruptcy is not easily predicted one way or the other.

The experiment involves a group of agents that retrieve data from multiple sources of information and use the Zeta Model to calculate the default risk of companies in a portfolio. Figure 4 shows a diagram of the conceptual system architecture.
Agent 1 is responsible for updating the portfolio information with updated calculations of default risk. Agents 2, 3 and 4 are responsible for retrieving information from their respective sources. Because they all share the same ontology, the agents are aware of the semantic differences between the information sources. For example, if Database 1 and Database 2 stored Total Liability values differently, one as “Total Liability” and the other “TL”, then when Agent 1 queries Agent 2 and 3, the semantic differences will be handled through the agents’ ontological underpinning.

For identification of concepts, we use the GT Guided Tool discussed previously. As the source data for the tool, we use language surrounding the Zeta model, excerpts from Wikipedia, and a textbook in economics [24]. The following excerpts are samples of input for illustration purposes.

Types of stock

Stock typically takes the form of shares of either common stock or preferred stock. As a unit of ownership, common stock typically carries voting rights that can be exercised in corporate decisions. Preferred stock differs from common stock in that it typically does not carry voting rights but is legally entitled to receive a certain level of dividend payments before any dividends can be issued to other shareholders. Convertible preferred stock is preferred stock that includes an option for the holder to convert the preferred shares into a fixed number of common shares, usually anytime after a predetermined date. Shares of such stock are called "convertible preferred shares" (or "convertible preference shares" in the UK)

Although there is a great deal of commonality between the stocks of different companies, each new equity issue can have legal clauses attached to it that make it dynamically different from the more general cases. Some shares of common stock may be issued without the typical voting rights being included, for instance, or some shares may have special rights unique to them and issued only to certain parties. Note that not all equity shares are the same.

Stock derivatives

For more details on this topic, see equity derivatives.

A stock derivative is any financial instrument which has a value that is dependent on the price of the underlying stock. Futures and options are the main types of derivatives on stocks. The underlying security may be a stock index or an individual firm's stock, e.g. single-stock futures.

Stock futures are contracts where the buyer is long, i.e., takes on the obligation to buy on the contract maturity date, and the seller is short, i.e., takes on the obligation to sell. Stock index futures are generally not delivered in the usual manner, but by cash settlement.

As a result of Ground Theory and the GT guided tool, the following terms came to be the essential concepts in this domain: Stock, Portfolio Order, MacroEvent Company, Profit, Loss, Buy, Sell, Order-Matching, Validate-Order, Validate-Portfolio, Take-Over, Market, Trader, Order-Type, Process-Order, Invalid-Order, Owns, Process-Order-Error, Amount, Price

Figure 5 presents a portion of the financial domain ontology where the concepts Portfolio, Stock, Company and MacroEvent are children of a root Concept. The single-barbed arrows indicate a parent/child relationship and the double-barbed lines between concepts represent concepts other relevant relationships. The concepts TakeOverEvent, LossEvent and ProfitDropEvent are child concepts of MacroEvent. For example, the relationships between the concepts Portfolio and Stock are 'containsStock' and the inverse 'isPartOfPortfolio'; the relationships between Company and Stock are 'isIssuedBy' and 'ownsStock', and the relationship between Company and MacroEvent are 'hasEvent' and 'eventBelongsTo'.

Figure 5: Financial domain ontology

Figure 6 shows a portion of the ontology for agent communication and collaboration as developed in Protégé. Four agents are included: EventAgent, TraderAgent, OrderProcessAgent, and OrderAgent, among which are shown the types of communications and underlying ontologies. The OrderAgent communicates a ProcessOrderRequest to an
OrderProcessAgent, where the ProcessOrderRequest appeals to the ontology for ProcessOrder.

Further, the trading behaviours of TraderAgents are as follows: (i) SimpleAgent exhibits primitive trading behaviour, (ii) IntermediateAgent has moderately informed trading behaviour, (iii) AdvancedAgent possesses sophisticated trading behaviour.

Each agent holds the same number and valuation of stocks. Each agent also holds a list of stocks that they are interested in buying. This list reflects real-world trading decisions to invest in technology stocks or blue chip stocks. For the purposes of performance comparison, the number of shares each agent buys or sells on each trade is equal.

The main differences in TraderAgent behaviour arise from an agent’s buying and selling strategies and from its reaction to market MacroEvents. For example, the SimpleAgent ignores trend indications given by market MacroEvents, while the IntermediateAgent and AdvancedAgent behaviours react to these events. Agents evaluate each conditional statement by consulting the financial ontology. The statements vary depending on the sophistication of each trader agent. For example, an IntermediateAgent incorporates the following conditions in its behaviour: (i) If a company has a loss, then suspend trading for time $T$, (ii) If a company drops in profit, then suspend trading for time $T$, (iii) If a company is being taken over, then buy shares, (iv) If a company currently holds takeover target company shares, then suspend trading for time $T$ and sell shares.

Each agent also has a trading portfolio comprising of realized and unrealized profit tables. These tables update dynamically upon completion of a successful transaction and are available at any time during the simulation.

Our goal in this work is not to discover new properties of dynamic markets, e.g. improvements to the Zeta model, but rather to test MOMA in creating such markets for study, ideally by domain experts with need for software development skills. To this end, the prior ontology development was useful in providing the requisite knowledge and framework for implementing the agents. The ontology development phase did not require programming; however some scripting was required to implement agent behaviours in the stock market simulation.

Agents need to merge information from diverse sources, e.g. peer agents, market data, and macro events. The financial ontology enabled agents to derive semantic understanding from the exchange of data. The advantage of being ontology-driven is that common domain knowledge can be specified in a single source, accessible to all entities. Thus, while the agents were heterogeneous in nature - having different behaviours, perceptions, and language, the financial world in which the functioned was consistent across all...
agents. Whenever an agent receives information from another agent - for example, when the Order Agent receives a sell order from a Trader Agent - no translation is required to understand the communication. Thus, the need for the financial domain information to be coded explicitly in agents is removed.

Table 1 shows the lessons learned from our study, including problems that were encountered for each step of MOMA, possible solutions, and recommendations. Table 2 shows our original objectives in this work and our evaluations thereof.

<table>
<thead>
<tr>
<th>Table 1: Lessons learned with MOMA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tasks</strong></td>
</tr>
<tr>
<td>Identifying and refining concepts and relationships to be used in the development of the ontology</td>
</tr>
<tr>
<td>Problems and Issues</td>
</tr>
<tr>
<td>The use of standard, off-the-shelf, high-level ontologies takes up more time than necessary because one has to sort through a large ontology for only a few concepts.</td>
</tr>
<tr>
<td>Solution/Recommendation</td>
</tr>
<tr>
<td>The developer must discern the time/effort required to build ontology from a standard ontology and the potential for re-use and modification. Recommend: Use the Grounded Theory approach to identify concepts.</td>
</tr>
<tr>
<td>Additional concepts were required for the ontology, e.g. buy and sell concepts have a TraderAction parent.</td>
</tr>
<tr>
<td>When testing using scenarios, the concepts that were missed were discovered. Recommend: Expect some amount of iteration in concept development.</td>
</tr>
<tr>
<td>Adding logic to applications with rules and axioms</td>
</tr>
<tr>
<td>It is not straightforward to add logic through rules and axioms to govern agent behaviours in an application, i.e. such behaviours to do not fall out for free during ontology development.</td>
</tr>
<tr>
<td>Work is needed to ease the complexity of adding application logic. Recommend: Follow standard software development practices with requirements, analysis, and design phases.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Evaluation of original objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
</tr>
<tr>
<td>1. Develop a generic, but domain-specific ontology for reuse and sharing among applications in a specific domain</td>
</tr>
<tr>
<td>Medium Success. The initial results are promising, but more testing is necessary to establish confirmation of this objective. The use of generic ontology such as SUMO is ill-advised for most domain-specific applications.</td>
</tr>
<tr>
<td>2. Move business logic and domain knowledge from underlying agent code to higher level ontology development</td>
</tr>
<tr>
<td>Medium-to-High Success. Domain knowledge can be moved from agent code to an ontology layer. However, some of the behaviour and business logic of the agents must be coded in during Agent Development phase</td>
</tr>
</tbody>
</table>
due to the fact that generation of code for axioms and rules is not supported.

3. Facilitate the use of tools to accelerate ontology-based application development

| High Success. MOMA is driven by the use of tools in a pipe-line as a part of its processes. The use of tools speed up the development for time consuming tasks as concept identification. |

4. Distinguish between the domain expert and agent developer, and allow more contribution from the latter in application development

| Medium Success. A distinction exists between the roles of domain expert and agent developer in MOMA and much of the work has been removed from the latter. However, the agent still must request information from the domain expert in order to create application logic. |

5. Ultimately enable complete application development by domain experts without the agent developer

| Low Success. Without agent developers, MOMA cannot produce a working agent application. The ultimate goal is to have the ontology and application logic be generated at the ontology level and then generate code that can be plugged into a generic agent framework. |

6. Summary

The framework and principles of multi-agent systems (MASs) are useful for developing distributed applications. However, the complexity of MASs require much low-level programming and software development skills in order to develop useful applications. Our goal in this work is to reduce the development effort by separating the domain ontology from agent development. As such, the paper described a phased, iterative design methodology for ontology-based multi-agent applications (MOMA). A complementary aspect of MOMA is a pipeline of select tools to execute MOMA, also described in the paper. MOMA is designed to be followed by domain experts and researchers without agent development and software engineering. Through the use of tools and driven by ontology, MOMA was applied to experiments in MAS development in the domain of financial services. The evaluation of the approach shows that the approach is partially successful.

7. Future Work

We have proposed a methodology to analyze, design and develop ontology-based Multi-Agent Systems; however, the methodology is not structured and generalized to be re-used in other domains. The work in [25] has developed a research model that can guide researchers and designers in applying domain knowledge and Information System Design Theory (ISDT) for Dual Information Systems (DIS) in order to construct a domain-specific ISDT. This particular work applied the model for creating a domain specific design theory for Dual Change Management Information Systems. A goal of our future work is to create an ISDT for Ontology-Based Multi-Agent Information Systems and then apply this theory in other domains such as healthcare and manufacturing.

Further, according to [26], the product components of the ISDT detail the meta-requirements, including all mandatory system features, that all information system instances must meet and the meta-design that all instances must follow. They will also include the most relevant kernel theories and prescribe the design product effectiveness hypotheses, i.e. the most central measurable benefits. The process component of an information systems design theory should describe the process for designing the system instances. Our future work will investigate these ISDT components.

The most important objective for this research is ultimately to enable complete application development by domain experts without the agent developer; however, this objective was only partially successful because code generation still required some tweaking by software developers. In future work we plan to compare and draw upon Domain Specific Languages and Domain Engineering which have been applied successfully in industry and which also support automated code generation from domain models in industrial scale. We recognize that perhaps the chosen paradigm is inadequate for solving our research problem. Other paradigms will be studied for better solutions.

Future work should improve upon MOMA by integrating the agent development phase with better tools. Ultimately, we envision MOMA as a useful methodology for developing ontology-based MASs that doesn’t require strong software development skills.

8. References


