xTAO: Enabling a Declarative Approach to the Specification of Multi-Agent Systems

Toacy C. Oliveira

PUC-RS Faculty of Informatics,
Av. Ipiranga, 6681 – 90619-900, Porto Alegre, RS, Brazil

toacy@inf.pucrs.br

Paulo Alencar,
Don Cowan

University of Waterloo,
School of Computer Science
200 University Avenue West,
Waterloo, ON N2L 3G1
Waterloo, Ontario, Canada
{palencar, dcowan}@
csg.uwaterloo.ca

Carlos Lucena

PUC-Rio, Computer Science Department,

Rua Marques de São Vicente, 225 - 22453-900, Rio de Janeiro, RJ, Brazil

lucena@inf.puc-rio.br

ABSTRACT

Research on software agents has produced a diversity of conceptual models for high-level abstract descriptions of multi-agent systems (MASs). However, it is still difficult and costly for designers that need a unique set of agent modeling features to either develop a new agent modeling language from scratch or undertake the task of modifying an existing language. In addition to the modeling itself, in both cases a significant effort need to be expended in building or adapting tools to support the language. An extensible agent modeling language is crucial to experimenting with and building tools for novel modeling constructs that arise from evolving research. Existing approaches typically support a basic set of modeling constructs very well, but adapt to others poorly. A declarative language such as XML and its supporting tools provides an ideal platform upon which to develop and extensible modeling language for multi-agent systems. In this paper we describe xTAO, an extensible agent modeling language, and also demonstrate its value in the context of a real-world application.

Keywords

Multi Agent System, Declarative Programming, XML

1. INTRODUCTION

Multi-agent systems (MASs) are seen as a type of systems composed of multiple autonomous components that share their capabilities to solve a problem dealing with decentralized data with no global system control [1,2]. In MASs agents interact with each other to reach their individual and shared goals. To model these systems, designers must have expressive modeling languages and tools to manipulate models expressed in those languages.

Many agent-based modeling approaches have been developed in academia and industry [3,4,5]. However, the relationship between agents and objects is not clear in these existing approaches. These representations often favor and capture one of the abstractions, but lack representation and semantic features to describe appropriately both agent and object abstractions [6]. To solve this problem, we have developed a conceptual framework called TAO in which agents and objects coexist and both agent and object-oriented abstractions and their relationship can be modeled [7].

However, although most agent modeling approaches share a set of fundamental modeling constructs and concepts, including agents, roles, goals, beliefs, and plans [8], a continuing proliferation of new agent modeling approaches and features is expected for several reasons. On one hand, there is continuous identification and experimentation with new agent abstractions and their combinations. In different domains there may be different concerns and depending on the purpose of the agent model, certain constructs may or may not be appropriate or useful. On another hand, there is still no consensus on the features that should be present in an agent modeling representation or on what these features should model. Therefore, an extensible agent modeling language is crucial to experimenting with and building tools for novel modeling constructs that arise from evolving research.

In this paper we describe xTAO, an extensible declarative approach to the specification of MAS, its supporting tools, and also demonstrate its value in the context of a real-world application. In contrast with previous approaches, our approach takes advantage both of the XML's large base of tool support and of its extensibility. Our approach allows more freedom to explore new possibilities and modeling techniques, while maximizing reuse of tools and modeling constructs.

This paper is structured as follows. In Section 2 we give an overview of our approach. In Section 3 we present the illustrative example and its TAO representation. Section 4 describes XML specification using a problem of consistency and interoperability among distributed databases and Web user interfaces. Section 5 illustrates the experimentation we made and finally Section 6 presents our conclusions and future work.

2. APPROACH OVERVIEW

xTAO models are used to specify MASs in a declarative way, allowing the representation of agent's characteristics such as beliefs, goals and plans. In xTAO, agents inhabit environments, can be grouped in organizations and can generate and perceive events.

To express the model declaratively, we have adopted the XML technology, since it has a broad support from standard bodies such as W3C and it is commonly accepted or adopted by the academic community [9, 10] and industry. Moreover, based on our past experience [11, 12], XML models are suitable for program manipulation.

xTAO is based on TAO, which is a conceptual framework designed to the MASs domain [7]. Since our representation deals with domain models, not meta-models, we need to instantiate TAO so that our abstractions can be represented in XML. In addition, we need to refine the original framework to accommodate specific needs of our representation. As illustrated in Figure 1, the development of our declarative approach consisted of three phases: the TAO Refinement, the xTAO specification and the Model Instantiation.

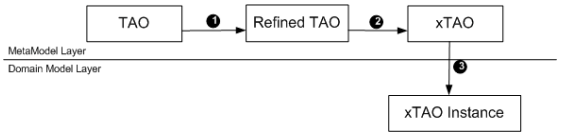

Figure 1: Development phases of our declarative approach

Phase 1 (Figure 1) deals with the refinement of the TAO Conceptual Framework so that more detailed information could be used when defining a TAO entity. For instance, we defined a structure called *Params* that must be used within a goal to represent the additional knowledge an agent must possess to attain such goal. In the same way, we defined a structure called *Description* that defines the data in a specific language such as a belief specification language [13].

The focus of Phase 2 is the specification of xTAO using a Document type Definition (DTD) [14], which is the XML meta-language. We have defined a refined-TAO vocabulary as a declarative programming language that provides the syntactic rules required to represent agents in the context of MAS. As a consequence, an instance of the Domain Model Layer is specified as a XML file that follows the DTD specification. The creation of such XML file is addressed in Phase 3 as the Model Instantiation and in our work it was used to validate and tune the TAO Refinement. The model was instantiated in the case of a multi-agent application dealing with consistency and monitoring problems that is described in [6] (see Section 3.2).

3. AN ILLUSTRATIVE PROBLEM

3.1. Overview

In this section we provide an overview of a Web application to monitoring data consistency and business processing based on declarative software agents [6] that we will use as our case study. In this approach declarative agent specifications that facilitate the use, programming and management of such agent-based systems allow for different distributed database and Web user interface structures to be interoperable. This declarative approach also relates business events to technical events while conveying to the users business meaningful events, that is agents that can deal with various levels of abstraction of a business object, and that can orchestrate and monitor data change and business processing.

Such Web application can be seen as a collection of high-level objects, low-level objects and events. By high-level objects we mean the objects that are semantically related to the representation of domain specific abstractions. On the other hand, low-level objects deal with the technicalities that define the execution of the application in the underlying computational environment. In this context, events emerge as the binding elements that provide the required stimulus to start a computational action. To achieve data consistency and monitoring in such scenario the semantic gap between the two levels of abstraction must be narrowed by entities that can perceive changes in the environment at both levels and can orchestrate the involved processes.

Agents are a natural abstraction to tackle this problem since this environment is highly dynamic and since agents can perceive events and execute actions that may depend on their internal states, rules or knowledge. Agents may also orchestrate actions related to a specific purpose by delegating some of their tasks to specialized agents that can help them.

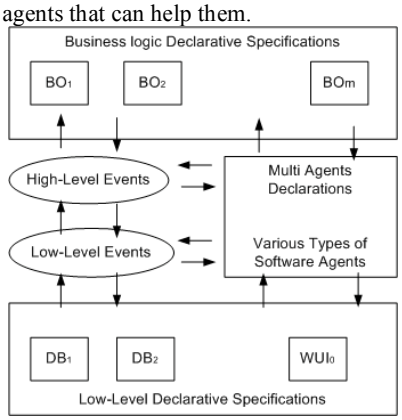

Figure 2: Overview of the data consistency and monitoring approach

In Figure 2, we present a Web application involving: distributed databases (and servers), Web user interfaces (and Web servers), events, business objects, and agents that can perceive events at various abstractions levels and can orchestrate the management process (e.g., refinement, consistency and monitoring).

The distributed databases (DB_1, …,DB_n) change constantly and should be kept consistent at all times. In most cases changes or additions to data require handling by people or processes within a larger system. Besides having an effect on other data, these changes may also affect business processes related to this data and be subject to business rules that govern the specific data sets.

Web user interfaces (WUI_o) may be defined by forms and other types of user presentation. These distributed user interfaces also need to be kept consistent. Sometimes changes or additions to these forms need to be handled within the context of a larger system. The changes in these Web interfaces may also affect the business processes related to the data forms and be subject to business rules that govern the presented data.

We define both low-level events and high-level events. In general, we want to define events that allow data content and business processes to be managed in a flexible way through agents that augment, refine, interconnect, ensure consistency, and monitor data and business processes. They can be related to the status of data and processes. Events can also be related to time and location. We also relate high-level to low-level events in order to know, for example, what low-level events somehow led to the occurrence of a high-level event, or what low-level events led to high-level failures in business processing (see Section 3.2).

A business object (BO_1, …,BO_m) is a representation of an active entity in the business domain, including at least its business name and definition, attributes, behavior, relationships and

constraints. A business object may represent, for example, a person, a place or concepts such as a domain entity or a user interface. The representation may be in a natural language, a modeling language, or a programming language.

3.2. The TAO representation

Our agent-based application deals with consistency and monitoring of Web user interfaces and Databases in a distributed scenario. By consistency and monitoring we mean the ability to ensure that elements in the same or in different levels of abstractions conform to a given structure. For example, we can check the integrity of the system's elements after a database maintenance procedure (update and/or addition) and guarantee the Web catalogs use accurate tables and fields names. In the same way we can provide a functionality that automatically fills a Web form based on the occurrence of a given event. To achieve such goals the TAO application instance must represent concepts such as databases, Web forms, services and events, and also represent ways to connect them.

During the development of this application instance we were able to identify some TAO entities that could be arranged in the configuration presented in Figure 3.

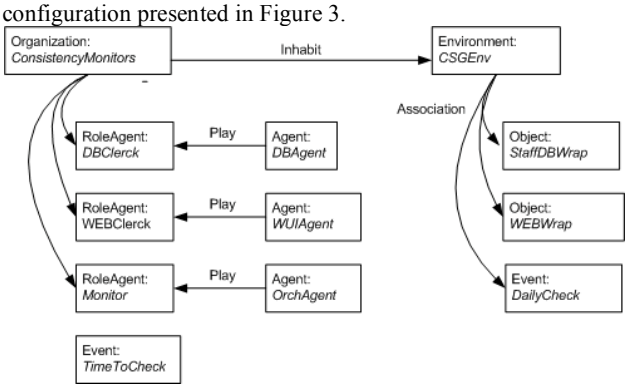

Figure 3: TAO entities related to the illustrative problem

The identified entities are:

Agent: *Agent is an autonomous, adaptative and interactive element that has a mental state (i.e., beliefs and goals).*

1. DBAgent - Database agents are used to connect to one or more databases (interoperability) and perform consistency operations on them. These agents perform tasks such as comparing and copying database content, indexing and harvesting Web content.
2. WUIAgent - These agents are used to connect one or more Web user interfaces (interoperability) and perform consistency operations among them. They can be used, for example, to check the consistency of two Web catalogs. These agents can perform operations such as comparing and copying the content of Web forms.
3. OrchAgent – These agents, which perceive low-level and high-level events, need to interact with other agents, and with low-level and high-level descriptions (e.g., the business-oriented declarations and the Web form and database declarations) in order to execute their actions. Agents that orchestrate the data and process change management needed to recognize events, decide what to do when these events happen, when, how and where to accomplish their tasks, and to whom they can delegate some of their tasks in order to accomplish their goals.

Object: *An object is a passive or reactive element that has state and behavior and can be related to other elements.*

1. StaffDBWarp - The StaffDBWrap is an object that encapsulates the access to the underlying Database system. It contains concepts such as connection, tables and fields.
2. WEBWrap – The WEBWrap is an object that encapsulates the access the Web environment. It contains concepts such as URL, forms and get (from HTTP).

Event: *Events are related to relevant changes in an entity state that can be perceived by other entities.*

1. DailyCheck – This event triggers a daily checking procedure in order to guarantee consistency.

Organization: *An organization is an element that groups agents, which play roles and have common goals. An organization hides intra-characteristics, properties and behaviors represented by agents inside it.*

1. ConsistencyMonitors – This organization is in charge of complex consistency checks performed by group of agents, which require a form of centralized control. This abstraction was used to accommodate the control relationship needed by our application.

Environment*: An environment is an element that is the habitat for agents, objects and organizations. An environment can be heterogeneous, dynamic, open, distributed and unpredictable* [15].

1. CSGEnv – The CSGEnv is the environment in which all the elements of the system reside.

Role: *Defined in the context of an organization, a role is an element that guides and restricts the behavior of an agent or an object in the organization. The social behavior of an agent is represented by its role in an organization.*

1. DBClerk – The DBClerk role is an agent role that provides access to the DB object to the organization.
2. WEBClerk - The WEBClerk role is an agent role that provides access to the WEB object to the organization.
3. Monitor – The Monitor object orchestrates the DBCleck and the WEBClerk in order to attain a more complex consistency goal.

4. xTAO SPECIFICATIONS

4.1. The Entities

According to TAO specification, a model consists of entities, each one defined by a set of properties. Entities represent first-order abstractions and have their semantics defined by the model. Properties related to the entities' state and behavior and are also defined in TAO. However the structure of each property, i.e., the bits and pieces that compose the information

conveyed by a property are not defined in detail and must be refined.

In this section we present the xTAO declarations, the adopted XML structures that were used to represent our agent-based application and examples. We also provide the rationale for refining TAO in order to accommodate our needs.

4.1.2. Agents

An agent is a TAO entity with properties defined in terms of beliefs, goals, plans, actions, events, roles and relationships [7]. Beliefs represent the agent knowledge about the world and in xTAO beliefs are denoted by the description tuple <*name*, *language*, *data*> where: the *name* field specifies the name of the belief that is internally used by the agent; the *language* field specifies the syntax and semantic that governs the information conveyed by *data*; and *data* is the information itself. The idea to adopt the description tuple is to provide an open representation of a belief, avoiding the definition of a belief language once this is not a goal of this work. Moreover this representation facilitates the use of any specification that follows a known grammar. As it will be seen in other cases, we have used this technique to represent most of the properties of TAO elements.

In Listing 1 we present the xTAO declaration for a belief that was used as part of a database agent specification. *DBAgent* goals are related to database operation. Thus, their beliefs must be aware of characteristics that facilitate reasoning in terms of database accessibility and manipulation. We've represented such belief using an object called *StaffDBWrap* that is responsible for database low-level manipulations.

```
<Belief>
  <Description>
    <Name>DBAcess</Name>
    <LanguageName>TAOLanguage</LanguageName>
    <Data> StaffDBWrap</Data >
  </Description>
</Belief>
```

Listing 1 – xTAO Belief declaration.

The other aspect that is important when defining an agent is its goal. A goal represents the intentions an agent possess in terms of desired state and in xTAO it is denoted by < name, desiredstate, params >. Once again the *name* field assigns an internal designation to the goal. The *desiredstate* is the most important aspect of a goal specification and represents the possible modifications of the agent state that are required in order to attain the goal. By agent state we mean all the knowledge that can be expressed using any aspect the agent is aware of, i.e., its beliefs, goals, plans, perceived and generated events and properties. *Params* describe the additional information needed to attain the goal.

Listing 2 presents the DBAgent *table_comparision* goal. The table comparison goal is defined to compare two tables and verify their similarity. After such goal is attained the agent knowledge (state) about the two tables (*srcTable* and *destTable*) involved must state if they were equal (EQ), not equal (NEQ), or if the comparison was unsuccessful due, for example, to a connection failure (NO). In order to represent the possible future states we have adopted the same description tuple used before but through a language form that allows us to use the xTAO elements that can influence the agent state. Thus, possible states can be represented as expressions such as *element1* AND *element2* where an element can be a goal, a belief, or an event. In Listing 2 it's possible to notice that the desired state *srcTable* EQ *destTable,* is represented in terms of the goal's parameters *srcTable* and *destTable*, the source and the destination tables subject to the comparison task.

We call this language the TAOLanguage and so far we have used elements such as beliefs, events and goal params[1].

```
<Goal>
   <Name>table_comparison</Name>
   <DesiredState>
     <Description>
       <Name>compareOK</Name>
       <LanguageName> TAOLanguage
       </LanguageName>
       <Data> srcTable EQ destTable |
              srcTable NEQ destTable | NO
       </Data>
     </Description>
   </DesiredState>
   <Params>
     <Description>
       <Name>srcTable</Name>
       <LanguageName/>
       <Data/>
     </Description>
     <Description>
       <Name>destTable</Name>
       <LanguageName/>
       <Data/>
     </Description>
   </Params>
</Goal>
```

Listing 2 - xTAO Goal declaration.

With this language it's also possible to specify more complex situations as the one found in our OrchAgent. OrchAgent controls a DBAgent and a WUIAgent in order to validate (check consistency) data and webforms. Thus, its desired stated is totally dependent on these two agent's goals. Listing 3 describes the consistency check goal's desired state in terms of the occurrence of the DailyCheck event plus the realization of DBAgent's table comparison and WUIAgent's data consistency.

```
<Data>
   DailyCheck AND
   DBAgent.table_comparison AND
   WUIAgent.data_consistency | NO
</Data>
```

Listing 3 - xTAO complex goal declaration.

The next step in the agent specification is the definition of its plans. Plans can be seen as the strategies used to attain a goal and in our approach they define the sequence of actions an agent executes. In Listing 4 we present an open-query-close plan that was adopted to achieve the table compare goal using the description tuple previously mentioned. So far, the match between goal and plan is established by comparing their two names (i.e., goal name and plan name), but we intend this process will be refined in later versions.

[1] Actually any abstraction of xTAO can be used to designate a future state but in this paper we have only used these three elements.

```
<Plan>
 <Description>
  <Name>table_comparison</Name>
  <LanguageName>TAOLanguage <LanguageName>
  <Data>open 1;open 2;query;close 1; close 2
  </Data>
 </Description>
</Plan>
```

Listing 4 - xTAO plan declaration.

Actions are related to agent's basic computations and are also represented by a description tuple and a parameter description. For example, the open action used by the DBAgent requires a table name to be passed and it is presented in Listing 5.
An important aspect of our approach is that it allows the use of any language to represent some properties. Thus the specification of an action can be done using, for example, C++ or Java. However, once we advocate a declarative approach, we prefer to use an action as a reference to an existing function provided by the environment or other entities.

```
<Action>
 <Description>
  <Name>open</Name>
  <LanguageName> External </LanguageName>
  <Data/>
 </Description>
 <Params>
  <Description>
   <Name>srcTable</Name>
   <LanguageName> External </LanguageName>
   <Data/>
  </Description>
 </Params>
</Action>
```

Listing 5 - xTAO action declaration.

Events indicate relevant changes of an entity state and can be seen as a gluing mechanism in terms of functionality. For example, our orchestration agent perceives the occurrence of the DailyCheck event that is triggered by the environment entity in start its checking procedure (See Listing 6).

```
<Events>
 <PerceivedEvent eventName="DailyCheck" />
</Events>
```

Listing 6 – xTAO perceived event declaration.

Relationships define the agent's knowledge of other entities and they will be addressed in Section 4.2.

4.1.3. Objects

In xTAO objects represent reactive entities and can be traced to the OO abstractions that are used to encapsulate operations (e.g., *behavior* TAG in Listing 7) and data (e.g., *state* TAG in Listing 7). However, once we have a declarative approach we do not attempt to fully specify the object's behavior, but we instead name the operations that can be used by other entities. Listing 7 presents the *StaffDBWrap* object specification in xTAO.

```
<ObjectClass className="StaffDBWrap">
 <State>
  <Information>
   <Description>
    <Name>connection</Name>
    <LanguageName/>
    <Data/>
   </Description>
  </Information>
 </State>
 <Behaviour>
  <Operations>
   <Description>
    <Name>Open</Name>
    <LanguageName/>
    <Data/>
   </Description>
  </Operations>
 </Behaviour>
 <Relationships/>
 <Events/>
</ObjectClass>
```

Listing 7 - xTAO object declaration.

4.1.4. Events

An event represents a relevant modification of an entity state that can be sensed by other entities. They can be used to initiate the execution of actions, as in objects, or be part of a desired state specification, as in agents. In xTAO events are represented as a simple name that holds the semantics of the event and the definitions of the parameters that actually represent the state that triggers the event. In Listing 8 we present an event that is triggered when the clock object reaches "3:00 am".

```
 <EventClass className="DailyCheck">
  <Params>
   <Description>
   <Name>time</Name>
    <LanguageName> TAOLanguage </LanguageName>
    <Data>TimeWrap.time EQ "3:00 am" </Data>
   </Description>
  </Params>
 </EventClass>
```

Listing 8 - xTAO generated event declaration.

4.1.5. Organizations

Organizations represent entities that are aligned with structures such as a hierarchy. As specified by TAO, xTAO organizations are represented in terms of rules, laws and relationships, and were not fully specified so far. In our example we used one organization to conform to some relationship polices dictated by the basic TAO model (see Section 3.2).

4.1.6. Environment

Environment is the entity that houses all other entities and is represented by its resources, services, behavior relationships and events. Resources can be seen as any real world entity that is used by the system and need to be somehow represented in it. In our case we used this abstraction to represent the Web and the underlying DBMS with the WEBWrap object and the StaffDBWrap object, respectively. Services hold the representation of public facilities [7] and were not used in our case. Behavior, relationships and events have the same meaning as in the agent and object abstractions. In Listing 9 we present the xTAO representation of the *CSGEnv* environment in our illustrative application.

```
<EnvironmentClass className="CSGEnv">
 <Resources>
  <Resource>
   <Object name="StaffDB"
    objectClassName ="StaffDBWrap"/>
```

```
  </Resource>
  <Resource>
   <Object name="FormsRepository"
    objectClassName ="WEBWrap"/>
  </Resource>
 </Resources>
 <Services/>
 <Behaviour/>
 <Relationships/>
 <Events>
  <GeneratedEvent eventName ="DailyCheck"/>
 </Events>
</EnvironmentClass>
```

Listing 9 - xTAO environment declaration.

4.1.7. Roles

Roles can be viewed as open spots defined within the context of organizations that are characterized by state, behavior and other properties. Agents and objects can fill these open spaces using the *play* relationship (See Section 4.2). xTAO defines roles in the same way of TAO and, once again, were used to conform to the basic TAO relationships restrictions.

4.2. The Relationships

A TAO model consists of entities that relate to each other by means of relationships. Although we have represented in xTAO all the eight types of relationships defined in TAO, we will focus our illustration on the ones that were used in our example: *Control* and *Play*.

The *Control* relation designates that a role A (the controlled) is controlled by a role B (the controller) and that B must satisfy A's demands. For example, in our application, we have specified an agent role called *Monitor* that uses the DBClerk and WEBClerk agent roles as assistants to achieve the consistency goal. As can be seen in Listing 10, in xTAO we represent this relation as a *control* tag that designates the name of the relationship (*MonitorDB*) and name of the controlled role (*DBClerk*).

```
<Relationship>
 <Control>
  <ControlRoleAgentRoleAgent relName ="MonitorDB"
   agentRoleName="DBClerk"/>
 </Control>
</Relationship>
<Relationship>
 <Control>
  <ControlRoleAgentRoleAgent
   relName ="MonitorWEB"
   agentRoleName="WEBClerk"/>
 </Control>
</Relationship>
```

Listing 10 - xTAO control relatioship declaration.

In the same way we define that the *DBAgent* plays the role of a *DBClerk* when associated to the *ConsistencyMonitors* organization.

```
<Play>
 <PlayAgentRoleAgent relName= "AgtoClerk"
  agentName ="DBAgent" agentRoleName ="DBClerk"/>
</Play>
```

Listing 11 - xTAO play relationship declaration.

5. DEVELOPMENT SUPPORT AND EXPERIMENTATION

Our multi-agent system approach was designed to be easy to use, program and maintain, and yet still offer a rich framework for creating and running multi-agent system. In the following we describe some features of our implementation and some additional applications we have developed to test our approaches.

In the context of our chosen application, we have defined XSL transformations to create C/C++ programs from XML agent declarations. As well, server side scripts have been set up to compile the C/C++ code into an executable image that can be downloaded back to the client system or saved on the server. The executable program (in Visual C) can be executed by other agents, or run from a command-line, from the web server, as part of a database trigger or in response to some other asynchronous event, such as a timed event.

The Computer Systems Group (CSG) at the University of Waterloo recently worked with several community groups in the Regional Municipality of Waterloo in Ontario, Canada to facilitate the presentation of local information throughout the community and beyond. In the project a web site was created to present a directory of employment support services in the region. The Waterloo-Wellington Training and Adjustment Board (WWTAB) had contracted with another local agency to contact local employment support agencies and to construct a database with an extensive description of each employment support agency.

WWTAB made the database of employment support agencies available to CSG as part of a community information project and agreed to maintain it. CSG provided the web site technology, database access services and site hosting facilities. Several focus groups were held around the community to evaluate the ease with which information could be accessed through the site. Because direct connectivity from the web server to the remote database has turned out to be too unreliable and slow to provide reasonable responses to web queries, a simple method for synchronizing the database content from WWTAB to the web server was needed. Both content comparison and copying agents were created in the initial implementation. On average, approximately one record per week is updated in the WWTAB database of about 400 records. The agent that was created was intended to synchronize the databases nightly.

Furthermore, WWTAB is one of 23 training and adjustment boards (TABs) in the Province of Ontario that are funded by the Government of Ontario to collect such information. Partway through the project, the province mandated that all of the local TABs must enter their data into a newly established central Web site for employment support. The contractor that had created the central database and web site did not permit any direct connections to their database – all local TABs must enter their data into web forms and each employment support agency record would require between 4 and 10 separate form pages to be entered. In response to this requirement, we have extended the XML/agent technology to include web form submission. This approach will also be used for ongoing maintenance as database records continue to change.

We are currently experimenting with other features of our agent modeling and implementation approach, such as more advanced consistency checking among distributed heterogeneous databases and Web user interfaces.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have described xTAO, an extensible agent modeling language, and its supporting tools, and also demonstrate its value in the context of a real-world application. We have also presented a specific application instance of our approach to illustrate some of its representative features, and also described an implementation and applications that have been developed as part of our feasibility studies.

As future work, we want to investigate the following issues. We want to improve our agent-based representations by incorporating definitions related to a more general specification for goals, a language that in this paper we called TAOLanguage. A challenge will be finding a suitable representation that enables the definition of the interdependency among agent goals, events, roles and beliefs.

Our envisaged xTAO development and support environment has four levels: description (agent and object abstractions), validation, transformation, and run-time platform. In this paper we have focused on providing a declarative approach to specify MAS. At the validation level, we will provide XSL translation to formal descriptions based on Prolog and process-calculus in order to validate agent interactions and behavior. At the transformation level, we will provide tools to transform our agent templates into programs in languages such as C++ or Java. At the run-time level, we have the execution environment.

We also want to use formal methods to define a formal specification notation and formal validation techniques that were sketched in this paper to establish certain attributes of the software agents and their interactions.

In order to make our approach more extensible, we are also working on translating our declarations to XML schemas [14]. XML schemas have a type system for XML that supports simple and compound types, plus a method for typed inheritance. This kind of type inheritance is absolutely crucial to creating extensible XML syntaxes. It allows users to define XML types that are composed of attributes and elements, and later define specific extensions to those types that can be used to add or remove attributes and elements from those base types, a technique similar to subtyping in object-oriented programming languages. The type extensions can be defined in a separate schema from the base type, a feature that allows developers to extend other developers' schemas without directly interacting with them.

In summary, xTAO allow experimentation with new modeling constructs and their manipulation without the need to rebuild and entire modeling language from scratch. We believe it can serve as a basis through additional extension for more expressive and comprehensive agent modeling representations.

BIBLIOGRAPHY

[1] Jennings, N.R., Sycara, K. and Wooldridge, M. A Roadmap of Agent Research and Development. In: *Autonomous Agents and Multi-Agent Systems Journal*, N.R. Jennings, K. Sycara and M. Georgeff (Eds.), Kluwer Academic Publishers, Boston, 1998, Volume 1, Issue 1, pages 7-38.

[2] Jennings, N.R. and Wooldridge, M. Intelligent Agents: Theory and Practice. In: *The Knowledge Engineering Review*, 1995, Volume 10, Number 2, pages 115-152.

[3] Wooldridge, M., Jennings, N., Kinny, D., The Gaia Methodology for Agent-Oriented Analysis and Design, Autonomous Agents and Multi-Agent Systems Journal, vol 3 number 3 pp. 285-312,. Kluwer Publisher, 2000

[4] AUML specification found at www.jamesodell.com/ExtendingUML.pdf

[5] F. Bellifemine, A. Poggi & G. Rimassi. "JADE: A FIPA-Compliant agent framework", Proc. Practical Applications of Intelligent Agents and Multi-Agents, April 1999, pp. 97-108.

[6] Alencar, P., Cowan, D.D., Mulholland, D.,Oliveira, T. ,Towards Monitored Data Consistency and Business Processing Based on Declarative Software Agents, Lecture Notes in Computer Science, vol. 2603, 2003.

[7] Silva, V., Garcia, A., Brandao, A., Chavez, C., Lucena, C., Alencar, P., Taming Agents and Objects in Software Engineering, Lecture Notes in Computer Science, vol. 2603, 2003.

[8] Green Paper specification found at www.objs.com/agent/agents_Green_Paper_v100.doc

[9] G. Cabri, L. Leonardi, F. Zambonelli : XRole: XML Roles for Agent Interaction Proceedings of the Third International Symposium "From Agent Theory to Agent Implementation" at the 16th European Meeting on Cybernetics and Systems Research, Vienna (A), April 2002.

[10] Dashofy, E.M., Hoek, A, Taylor, R.N., An infrastructure for the rapid development of XML-based architecture description languages, Proceedings of the 24th international conference on Software engineering, 2002, pp. 266-276,Orlando, Florida, http://doi.acm.org/10.1145/581339.581374, ACM Press.

[11] Oliveira, T.C., Alencar, P., Cowan, D. : Towards a declarative approach to framework instantiation Proceedings of the 1st Workshop on Declarative Meta-Programming (DMP-2002), September 2002,Edinburgh, Scotland, pp 5-9.

[12] Oliveira, T ,A Systematic Approach to Object Oriented Framework Instantiation, PhD Thesis, Computer Science Department, PUC-Rio, Brazil, 2001.

[13] Shapiro - Steven Shapiro, Yves Lesperance, Hector J. Levesque The Cognitive Agents Specification Language and Verification Environment for Multiagent Systems, AAMAS-02, 2002.

[14] XML technology specification found at www.w3c.com

[15] Omicini, A.: SODA: Societies and Infrastructure in the Alalysis of Design of Agent-based Systems. In Ciacarini, P., Wooldridge, M. (eds) Agent-Oriented Software Engineering, Springer-Verlag 2001, pp. 185-194.