Adaptive Agent Model: an agent interaction and computation model

Abstract

Software systems must be capable of coping with continuous requirements changes and at the same time wisely make use of emerging components and services to remain useful in their environment. In this paper, the Adaptive Agent Model (AAM) approach is proposed. The AAM uses configurable interaction models to drive adaptive agent behaviour. The models capture user requirements and are maintained by experts at a high level of abstraction. The AAM interaction model has been discussed with regard to interaction specification and interaction coordination, in line with a coordination language for the OpenKnowledge project. A major benefit of using the approach is agents can dynamically choose the right partners for interaction, and the appropriate components and services for computation at runtime, when a new interaction model has been configured for them towards an emerging business goal. A simple expert seeking scenario has been used to illustrate the approach.

1. Introduction

Business environments and business needs are changing constantly and rapidly. Supporting software must change if it is to remain useful in the changing environments and maintaining customer satisfaction. The Object Management Group's (OMG) *Model Driven Architecture* (MDA) [6][7] promotes the production of models with sufficient details that they can be used to generate or be transformed into executable software [10]. It proposes a *Platform Independent Model* (PIM), a highly abstracted model, independent of any implementation technology. This is translated to one or more *Platform Specific Models* (PSM) based on a particular technological implementation (e.g. specific constructs, features of the implementation). PSM is translated into code in a similar pattern. The automatic translation process will rely largely on the UML 2.0 initiative, which facilitates software modelling with more precise semantics than its previous versions [9]. *Action Semantics* [11], for example, can be used to support the definition of actions in an executable UML behavioural model. Consequently, models with domain requirements captured can be turned into working code more reliably and more quickly [8].

However, as business acquisition and dynamic collaboration is the norm and responsive evolution is the essence today, standalone model building and rebuilding from scratch is becoming expensive and sometimes impossible. A system usually needs to integrate many different components and services readily available from a variety of sources, full specifications of them being often inaccessible but the reuse and interoperation of them through their published interfaces being predominant. Such a situation changes the "requirements first" perspective and a *Construction by Configuration* (CbC) [3] [4] approach becomes appropriate, where what the system should do is satisfied by a set of existing components and glue code that links them. For example, a generic *commercial off-the-shelf* (COTS) package developed for hospitals in England has been adapted for a patient management system used in Edinburgh, Scotland under the paradigm [5]. This implies that software requirements depend not only on what stakeholders believe they need but also what capabilities components already in place can provide. Hence Requirements Engineering is tightly integrated with implementation (Integrated Requirements Engineering [3]).

The CbC approach, however, also has its limitation, in its reuse and reconfiguration of components for across-domain applications. More importantly, when sources of components and services that CbC attempts to integrate into its single operating architecture are distributed over the internet, we are confronted with difficulty in coordination. For example, a flight booking agent might have to request a remote web service provided by a credit card company to authenticate a customer payment, before it could issue an e-ticket by invoking a local component. The replacement of components and services as well as their links that become obsolete over time can be explicitly configured with CbC, being much easier than the traditional means of recoding the hard-coded components, services, and their collaboration. Nevertheless, the change of glue code will interrupt the ongoing businesses. Moreover, components and services distributed over various sites in the environment could be designed for a wide range of purposes, have pre-determined capabilities, and expose incompatible input/output formats. Constructing a system by configuration requires matching the published capabilities of locally maintained components against the ever changing requirements and letting them dynamically join in and contribute to the interaction. Not all emerging interactions being predictable, the selection and coordination of the best collaborative candidates is a complicated task and determined to be error-prone if done manually. Therefore, a high level interaction model of the system must be developed in place of glue code and maintained at the business level in order to guide the low level capabilities-to-requirements matching. The *OpenKnowledge (OK)* [2] project has recognised the importance of such a model and its use of a comparable

yet more advanced architecture over CbC is discussed in this paper.

To conclude, MDA can be viewed as requirements-to-capabilities generation while CbC can be viewed as capabilities-to-requirements matching. The former approach has not taken into account the need of dynamic interoperation of disparate components and services in the system under development. This prevents full reuse of existing infrastructures. The later has ignored a broader interaction model of components and services within the system. Thus it demands extra glue code management. We put forward the *Adaptive Agent Model* (AAM) approach that encompasses advantages of both described approaches but avoids their limitations.

2. The Adaptive Agent Model and OpenKnowledge

The Adaptive Agent Model has been developed for large business applications to cope with changing business requirements and to ease the continuous maintenance of supporting software [12]. Briefly, AAM is a methodology that guides the building of an organised hierarchy of business knowledge models [22] to drive adaptive agent system behaviour. The models originate from business requirements, are interpreted/executed by agents at runtime, and are under continuous maintenance by business people. Tools have been developed to support the documentation and maintenance of models. Existing object-oriented infrastructures can be reused to support agents to execute business requirements captured in models. Rule is used as a central AAM model element. They are directly derived from functional requirements specification [12].

The AAM approach was intended to add adaptivity to single large object-oriented software systems. However, it can be easily adapted for use in distributed environments such as *OpenKnowledge* [2]. The tailored rule-based interaction models in AAM are reconfigurable and agents interpret their behaviour dynamically from them (requirements-to-capabilities). Moreover, agents as high level abstractions, make use of a combination of exiting components and services to meet their required functionalities (capabilities-to-requirements). The running AAM system is distributed heterogeneously.

In a heterogeneous environment, numerous constructs of a variety of types (objects, services, and agents) are available via independent development. Alternatives from competitive service providers can be explored for (re)use towards evolving business purposes. Apart from (1) component/service/agent providers and (2) end users, a third important role might be played in such a system by interaction model designers. They have application domain knowledge, choose components from alternatives, and specify their interactions to fulfil business goals.

Although standard languages exist for describing component interfaces or defining interactive message passing protocols, for instance, DAML-S for Web Services and FIPA ACL and KQML for MAS, their usage is limited in their specific areas. So far no mechanism for coordinating and interoperating disparate components has been provided. The difficulty becomes more severe if selecting the right components and specifying cooperation towards an emerging business goal must be accomplished in a timely way and dynamically. The OpenKnowledge (OK) project [2] proposes a new form of knowledge sharing based on declarative interaction, the specification of which can be transmitted and interpreted by peers involved in interaction at runtime. Each peer appearing in the OK system is equally important, having individual capabilities, and being able to be automatically involved in emerging interactions according to their semantic description. The major advantage of the proposed framework is that already established software paradigms are extended and integrated such that their interaction becomes automatic and the change of the interaction only involves the change of the interaction specification.

Components in the OK system cannot interact with each other automatically simply by using descriptions of individual components coupled towards a given goal. The acquisition of two types of knowledge is vital in the proposed framework. An Interaction Specification controls the message passing from one component to another. This is done by associating their input and output in the control flow and by the components collectively contributing to the eventual goal of business. Interaction Coordination provides a form of coordination among components so that, when a component is found to be useful it can fit into the interaction with matching input and output, as expected by other components through the coordination. Note that all components have already been developed prior to the emerging interaction.

Interaction Specification and Interaction Coordination should be dynamically defined and amended (possibly at runtime) by interaction model designers. This is based on knowledge about the business domain and employment of the available components currently provided by component providers or developers. The exchange of employed components in interactions or the redefinition of interactions brings dynamic effects transparent to end users. New components can be freely added to the system and three roles, namely, end users, interaction model designers, and component providers can freely join the system.

OpenKnowledge uses the Lightweight Coordination Calculus (LCC) language to specify agent interaction and role playing processes. LCC is a simple design language for expressing interaction protocols. A primary aim of

LCC is to interfere as little as possible with the design and operation of individual agents [16]. LCC based interaction model design is done by knowledge engineers, who have skills in logic programming. The direct use of logic has its advantages for research purposes but in practice it needs to be associated with a more accessible interchange language [13]. For real business applications, business people (possibly being interaction model designers) want to control their own businesses opportunistically at runtime. The specification of how businesses operate should be accessible to the wide range of software engineers and preferably to business people. The AAM modelling approach complements LCC in this aspect as we show in the next section, where UML and XML style model declaration and message exchange is enabled. Business-oriented tool support is described in [12].

Another major advantage of using AAM to accompany LCC lies in its added adaptivity that the current LCC protocols need [13]. Replacement of a section of protocol by an individual agent with another section within an interaction when certain business conditions change is difficult for LCC at the moment but inherently available in AAM. The interaction model specification and coordination of AAM is friendly to interaction model designers and highly adaptive to agents. We discuss the AAM models along with the complementary LCC protocols. A simple example adapted from the one in the OpenKnowledge manifesto [2] is used for illustration.

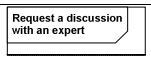
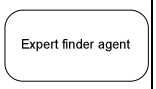
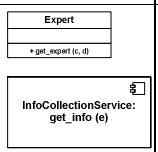
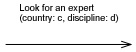
3. The AAM approach

Example: Suppose someone in the University of Southampton wants to find an expert on multimedia annotation in the UK. If the expert is found to be local in the university, a meeting will be attempted. Otherwise the expert will be contacted by telephone. Suppose also all UK expert information has been registered nationwide in a database that can be accessed by a component available in the system. By passing the country (UK) and discipline (multimedia annotation) of the expert, the name of the expert will be obtained. If multiple experts are found, the first one is considered. Additionally, a Web Service that returns the address and telephone number of a person (assuming the name is known) has been provided.

3.1 Notions and Notations

The AAM framework built upon the notions and notations outlined in Table 1 has a three-layered architecture. The first layer consists of agents interacting with one another by passing messages, using the behavioural knowledge from the next layer. The middle layer is a structured knowledgebase of rules, supplying to agents from the previous layer and referring to the components from the next layer. The last layer consists of computational units, ready to be invoked to facilitate the execution of the interaction model. The knowledge in the middle layer is expected to be updated continuously during the running of the system corresponding to changing requirements at runtime.

Table 1. Notions and notations of the AAM

Interaction model		A protocol model that describes the interaction process of multiple agents aimed at a common goal.
Agent		A high level abstraction that conceptually has common goals shared with other agents and computationally has responsibilities for contributing to the goals. Rules are defined that decide the roles an agent should play in a certain interaction aimed at a certain goal and the low level computational units should be invoked in the process. Object components, Web Services, and even other agents can all be used by an agent. Agents interact with one another by passing messages, the processing and producing of which is also determined by rules.
Rule		A requirement capture unit that externalises agent knowledge and is configurable at runtime by domain experts. Agents use rules to understand and respond to messages, make decisions, and collaborate with each other. A collection of rules compose and define agent interaction models. Various rules can be defined for a single agent to play different roles in different interactions models.
Object component and Web Service		Traditional passive components that respond to active agents when they are invoked, assisting the running agents to behave. The invocation of these low level units is defined in rules.
Message		An information container passing between agents. Messages are known by agents that create them and are expected by agents that receive them, if related rules are defined. String, XML fraction or even objects can be encoded in them. The passing of a message indicates the sender has made its contribution towards a business goal and now the receiver takes its responsibility to contribute to the same overall goal.

Having defined the AAM framework of its notions and notations, we start the discussion of Interaction Specification and Interaction Coordination with regard to the given example.

3.2 Interaction Specification

Figure 1 specifies an interaction model. It describes an interaction towards a goal; the required agents and their associated rules (which decide role playing; component

and service invocation; as well as message passing). The specification of rules shapes the control structure of the interaction. They should be made configurable by human and executable by agents. R2 has its notion and notation introduced as a rule in general in Table 1. The specification of this sample rule is given in Figure 2 and it is assumed that Expert. get_expert (c, d) is a published component interface.

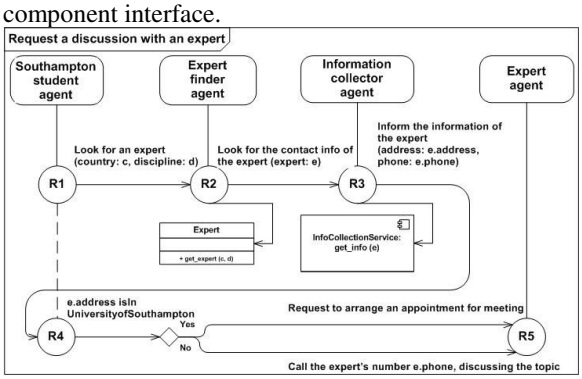


Figure 1. The Interaction Specification of the example

Rule R2
Owner Agent:
Expert finder agent
Use Context:
Interaction model: Request a discussion with an expert
Role Description:
Look for the name of an expert (The rule decides its owner agent should play this particular role in the interaction model it is currently involved)
Role Playing:
Triggering Event:
Respond to: Southampton student agent
Incoming message: Look for an expert (country: c, discipline: d)
Processing:
Component use: Expert
Invocation call: Expert. get_expert (c, d)
Return result: Expert e
Further Action:
Request to: Information collector agent
Outgoing message: Look for the contact info of the expert (expert: e)

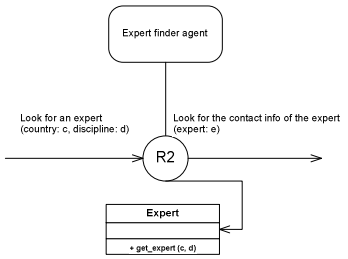


Figure 2. The specification of a rule

In the interest of conciseness, only the fundamental rule modelling feature is outlined in this paper. We allow in the specification, among other additional constructs, multiple {condition, action} couplets, where different actions can be taken in different corresponding conditions. That is useful for R4 in the example and the full description of the framework can be found in [12]. The principal aim of the AAM is not to propose yet another modelling language (for example UML) for a specific programming language. Rather, AAM generically describes the dynamic integration and interoperation of disparate components and services in evolving systems. The following clauses express the models of the same interaction in the OpenKnowledge LCC language.

a(R1(c,d,s),SSA)::
lookup_expert(c,d) ⇒ a(R2,EFA) then
contact(c,d,a,p) ⇐ a(R3,ICA) then
a(R4(a,p,s),SSA)

a(R2, EFA)::
lookup_expert(c,d) ⇐ a(R1,(c,d,s),SSA) then
lookup_contact(c,d,e) ⇒ a(R3,ICA) ←get_expert(Expert,c,d,e)

a(R3, ICA)::
lookup_contact(c,d,e) ⇐ a(R2,EFA) then
contact(c,d,a,p) ⇒ a(R1,(c,d,s),SSA)
←get_info(InfoCollectionService,e,a,p)

a(R4(a,p,s),SSA)::
request_appointment(a) ⇒ a(R5,EA)←a∈s or
call(p) ⇒ a(R5,EA) ←a∉s

Figure 3. LCC clauses for the same interaction

Briefly, $a(R_i, A_i)$ :: Def denotes that an agent (type) A_i plays a role R_i as defined in Def. Def describes the message passing behaviour constructed using the following forms: Def_j *then* Def_k (Def_j satisfied before Def_k), Def_j *or* Def_k (either Def_j or Def_k satisfied), or Def_j *par* Def_k (both Def_j and Def_k satisfied). In the Def, $M_l \Rightarrow A_m$ denotes that a message M_l is sent to agent A_m while $M_l \Leftarrow A_m$ denotes that a message M_l is received from agent A_m. Also in the Def, $\leftarrow Cons_n$ denotes that a constraint must be satisfied before the clause prior to it. For a full LCC dialogue framework description, please refer to: [13] [16].

The explanation of the definitions of R1, R2, R3, and R4 in LCC for our example using this framework is as follows. The Southampton student agent (SSA) initially plays the role R1 by sending a lookup expert message with country c and discipline d as parameters to the Expert finder agent (EFA). SSA is now waiting for a message from the ICA. In the meantime, on receipt of the message from the SSA, EFA responds by playing role R2. It obtains an expert with his/her name by invoking a get_expert() method of an Expert component, passing the c and d parameters from SSA. The expert e along with the original c and d parameters are encoded in a lookup contact message, sending to the Information collector agent (ICA). In a similar means, the ICA invokes an InfoCollectionService to get the address and phone number of the expert, passing them to the SSA. On receipt of the message with contact information, SSA becomes active again and will change its role from R1 to R4. That role tells the SSA to request an appointment with the expert if he is within the range of Southampton University; otherwise the SSA will make a phone call to the expert in a remote site.

As illustrated above, AAM rule definitions and LCC clauses can equally express interactive message passing behaviour as associated with agent roles. An agent role playing behaviour within the context of a particular interaction model (as shown in the second LCC clause in Figure 3) can be derived from a rule specification (as shown in Figure 2). The triggering event part of the rule determines the incoming message passing pattern in LCC; the processing part of the rule determines the constraint solving pattern in LCC; and the further action part of the rule determines the outgoing message passing pattern in LCC. The overall reaction pattern and the constraint solving involved in it structure the AAM rule and LCC represents the rule construct in its logic expression.

The LCC language itself, however, does not assume any specific mechanism for agents to judge constraint satisfaction which is left up to the local agent implementation. In contrast, this computational model is directly specified by external component/service usage in integrated rules. They explicitly tell agents which computational units they should invoke in interactions. Knowledge is maintained in rules and will be changed when, for example, new services become available. Coordination of computation in matching inter-agent input/output is also the responsibility of rules (detailed in Section 3.3). LCC and its variants, as a form of logic programming language, have received support from analytical techniques such as model checking [17]. Rule-oriented AAM models ease agent interaction role configuration and alternative component/service selection in agent-oriented UML models. The two languages thus are complementary.

3.3 Interaction Coordination

An Interaction Specification specifies a pattern through which agents interact. What's equally important is the coordination of the involved computational units, in our example, the component "Expert" and service "InfoCollectionService", selected by R2 and R3 for EFA and ICA in their interaction. Agents can be developed from multiple agent platforms and components they use can be written in a range of languages, the collaboration of which is unknown to their original designers. We must coordinate the message passing behaviour, so that the output of one agent resulting from the computation of one component can be useful as the input of another agent, where another component that agent uses can take the previous computation result in an appropriate form for its own computation. If different languages are spoken, then ontology matching might be required. Ontology recognition and translation among interactive agents being an engineering issue for constraint solving as associated with the LCC dialogue framework, the matching of component/service input/output is identically crucial for AAM interaction coordination or any other knowledge interchange system. Though ontology matching is not the main topic of the paper, the following provides a scheme of how AAM as a technology-independent approach, can support the coordination.

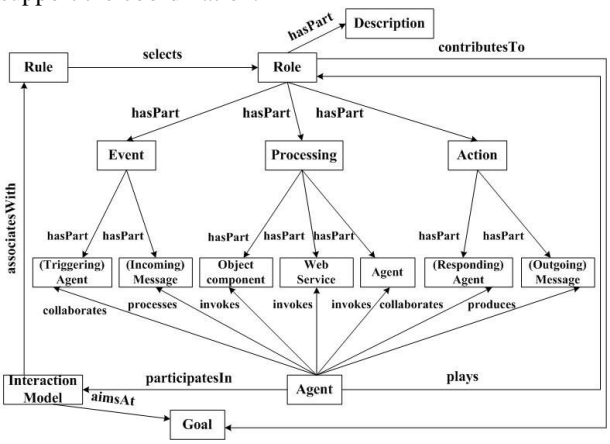


Figure 4. Domain-independent ontology

The conceptual model of Figure 4 defines a domain-independent ontology that consists of a set of cross-domain reusable language terms that all agents shall use to act upon interaction specifications. Encapsulated in event and action messages are domain-specific ontology, instances of which are populated in agent conversation while agents communicate at runtime. The defined graph provides an envelope in which any domain specific contents can be used in conversation. Only agents in a specific domain are associated with the related domain ontology. This allows an extensible ontology to be later developed for agents assigned to a new domain and its application using the existing agent architecture. This requires a minimum set of common agent knowledge and the core agent architecture becomes easy to maintain.

The UML style of AAM interaction models is friendly to human experts for reading and configuration. They also need to be in a form suitable for agents to interpret and execute, upon disparately designed components and services. Based on the two sets of ontology, XML-based rule definitions are used in our AAM models to support model interpretation and execution. These are: 1) suitable for machine processing; 2) unique for message interchange across platforms; 3) enabling for interoperation. Figure 5 shows such a rule definition for our example. XML schema corresponding to the conceptual model is omitted due to the space limit but can be found in [23].

In a business environment, the established CORBA and IDL architecture [6] enables interoperation by compiling exchange message formats into programs. In order to permit the exchange of structured data only known locally at runtime in distributed systems, a typical method is described in [20]. This involves: (1) discovering the

metadata for message transmission; (2) binding of program objects to the metadata; (3) marshalling and unmarshalling of data. AAM rules facilitate the coordination of components and services without extra components (discovery and so on) or processes (any process other than the rule processing). When agent EFA and ICA interact using R2 and R3 respectively, a shared schema is specified in the action message of R2 and the event message of R3. Both agents expect this schema so that they can exchange information without external translation. In this case, it is a lookup contact message with expert first name element supplied before last name element. During the processing of rules, the internal representations will be converted to the XML structures according to the common schema and vice versa. The modifiable nature of rules allows, when component interfaces change, the reconfiguration of exchange message format between a pair of agents that start using the new format immediately.

```
– <rule>
    <name>R2</name>
    <interaction-protocol>
      Request a discussion with an expert
    </interaction-protocol>
    <owner-agent>Expert finder agent</owner-agent>
  – <global-variable>
    – <var>
        <name>c</name>
        <type>Country</type>
      </var>
    – <var>
        <name>d</name>
        <type>Discipline</type>
      </var>
    – <var>
        <name>e</name>
        <type>Expert</type>
      </var>
    </global-variable>
  – <event>
    – <message>
        <from>SSA.R1</from>
      – <content>
        – <lookup_expert>
            <country>c</country>
            <discipline>d</discipline >
          </lookup_expert >
        </content>
      </message>
    </event>
    <processing>
      e = Expert.get_expert(c,d)
    </processing>
  – <action>
    – <message>
        <to>ICA.R3</to>
      – <content>
        – <lookup_contact>
          – <expert>
              <first_name>e.first_name</first_name>
              <last_name>e.last_name</last_name>
            <expert>
          </lookup_contact >
        </content>
      </message>
    </action>
    <priority>5</priority>
  </rule>
```

Figure 5. Rule specification in XML

For a detailed explanation of rule execution process by agents, please refer to [12]. That literature also provides AAM deployment architecture. Pseudo code of the example rule being evaluated, interpreted and executed by an agent deployed by AAM is shown in Figure 6.

```
thisAgent. addBehaviour (Rule thisRule) {
thisBehaviour. setPriority (thisRule. getPriority ());
    Country c;
    Discipline d;
    Message m = thisAgent. receiveMessage ();
    while (m != null)
    {
      Agent fromAgent = m. getSenderAgent ();
      if (fromAgent. equals
        (thisRule. getEvent (). getMessage (). getFromAgent ()))
      {
        XMLSchema schemaIn =
          thisRule. getEvent (). getMessage (). getSchema ();
        XMLSchema schemaOut =
          thisRule. getAction (). getMessage (). getSchema ();
        ObjMsg lookup_expert =
          m. getContentObject (). unmarshal (schemaIn);
        c = (Country) lookup_expert. getCountry ();
        d = (Discipline) lookup_expert. getDiscipline ();
        Expert e = Expert. get_expert (c, d);
        if (e != null)
        {
          XMLMsg lookup_contact = e. marshal (schemaOut);
          Message m2 = new Message ();
          m2. setContentObject (XMLMsg);
          Agent toAgent =
            thisRule. getAction (). getMessage (). getToAgent ();
          m2. addReceiverAgent (toAgent);
          thisAgent. send (m2);
        }
      }
     m = thisAgent. receiveMessage ();
    }
}
```

Figure 6. Pseudo code of an agent behaviour interpreted from an XML-based rule

The XML schema based message matching contrasts with vocabulary based ontology matching, AAM agents running on different platforms being capable of exchanging complex annotated messages by encoding/decoding local objects, apart from simple strings. Traditionally, objects are written in the same language for mutual communication and understanding. For example, in JADE [19], the following code might be used to pass a List object, whose structure must be understood both by the sender and the receiver to enable their communication.

```
public void passList (List l) {
  ……
  // Fill the message content with a List l, containing the object
  fillContent(requestMsg, l);
```

AAM messages are platform independent and reusing existing messaging systems is easy. XML-based information can be used as object representation to be filled in or extracted from FIPA Agent Communication Language (ACL) [1] message contents. Hence, there is no requirement for the use of the same underlying object language for agents in different systems to understand each other. Data binding techniques such as Java & XML data binding [18] can be applied to convert between Java object instances and XML instances. Similar bindings can be applied to other programmed objects. A process called unmarshalling while receiving a message is for the XML data structure to be populated into member attributes of

the object, instantiated according to the XML schema just like from its class. Conversely, a process producing XML instances from objects is called marshalling while sending a message. XML is used as the interchange medium for components written in different languages. Therefore, a list object in Java can be marshalled by an agent to XML, transmitted over the network, and unmarshalled as a C++ object understood by another agent for its internal use.

Apart from the platform-independent message passing, AAM is also agent behaviour implementation neutral. For example, the current practice of defining agent behaviour in JADE is by specifying behaviour methods in agent classes. Built upon the established platforms and conventions, AAM agents understand the domain-independent ontology defined in Figure 4 and can use a simple ontology to agent behavioural construct mapping to execute rules [23]. When an incoming message arrives at an agent, it retrieves a set of relevant rules and selects the appropriate one that matches with the event to be dealt with (by schema-matching), processes it and then sends an outgoing message. This whole process can be coded as an ordinary agent behaviour method but in effect it is dynamic according to the description of the rule currently selected in that context at execution time. In sum, the AAM approach is applicable to any (or a combination of) existing software infrastructure and adds adaptivity.

4. Conclusions and Future Work

Metadata has long been used as a mechanism by the Software Engineering community to reduce duplicated code, alleviate programming tasks, or even automatically generate systems [15]. For example, metadata describing class structure, property, and behaviour is used by the Adaptive Object Model approach to generate object-oriented systems [14]. In general, the idea is, when requirements changes arrive, the changes captured in metadata can be transformed to the software systems. This is the driving force of the OMG's MDA methodology. Though metadata is useful in these approaches, when manual code change is necessary because metadata cannot cope, such change to the generated system is lost the next time the system is to be re-generated. More importantly, generation is only possible when all metadata about the system is in place where the generation is performed. This assumption does not hold for distributed heterogeneous systems, where components and services are maintained in remote sites but they need to be composed dynamically. In addition, the generation of the running system presume all composing constructs are in the same language for the generator. Cross platform and language component and service reuse is not supported by the mechanism.

The traditional value of metadata is inherited by our rule-driven agent-oriented system. Rules capture user requirements, describe how components and services from various sources are coordinated by interaction model designers, and drive running agent system behaviour dynamically on the fly. The use of this enhanced metadata shortens requirements-design-implementation software development life cycle within our integrated framework. Its auto-interpretation rather than auto-generation nature guarantees the minimum maintenance efforts. The continuous maintenance of knowledge about interaction models rather than code is emphasised. It is business people who are responsible to maintain their system models in an explicit and visible manner. The maintenance of the model is eventually the maintenance of the software system.

A domain-independent conceptual model (Figure 4 including meta-ontology) directs agent behaviour, the interpretation of which forms agent roles individually and interaction models globally. A domain-dependent conceptual model (including domain-ontology) determines agent conversation contents, domain entities and their associated properties, the interpretation of which forming agent knowledge about tasks to be performed via web service and component invocation. Consisting of these two sets of ontology, AAM rules provide a context-aware agent framework, in which when/what components and services that should be used in given conditions are captured in context. The notion of context in the physical sense [21] (user location dependent communication service provision, etc.) can be borrowed here and adapted as in the conceptual sense (user need sensitive service provision, etc.). Nevertheless, this work shares with the context-oriented research trend some common characteristics: context-based service association, action triggering, and adaptation. Further development on defining context sensitive agent behaviour upon a more comprehensive ontology will be carried out in our OK project.

Although full automation is the ultimate goal, we at present emphasis human manageable models that make use of existing components in an adaptive manner. When interaction models are specified, alternative components that support model execution might be discovered automatically and selected according to a ranking/recommendation system. Though this helps the automation of interaction process, some kind of human control must accompany it. For example, predictable behaviour is vital in biomedical domains such as *HealthAgents* [24]. Successful completion of a dialogue for a given purpose cannot always be guaranteed in a dynamic protocol passing and invitation manner if all supporting services are automatically selected and executed. Unreliable interaction results in this context could possibly cause tragedy. Nevertheless, our approach does not prohibit the semi-automatic mechanism where a list of appropriate components/services presented to the

user interface for composition of interaction model using these underneath facilitating component invocation after human validation. This improves working efficiency: interaction model designers do not need to lookup all available services to any situation. This is also useful to extend existing models for related purposes, e.g. the replacement of example service with a service that can find all national addresses enables to look for anybody to do anything.

AAM is a non-invasive approach, its introduction of an agent abstraction system requiring no change to existing components and services previously designed. Any agent could dynamically contact any component at runtime to accomplish their roles, which is in contrast with agentification approaches, where transducer, wrapper or rewrite is imposed upon the given components for a one-to-one conversion. The most distinctive characteristics of both AAM rule modelling language and LCC language is specifying interactions towards a given goal at runtime, when participating agents are unaware of such an interaction and of the roles they shall play until the model is required to be executed. New requirements can always be guaranteed to be deployed once they are specified, as both interaction and computation in our MAS are adaptive. We believe the visual modelling of AAM and concise syntax of LCC can work together for mutual advantages. For example, while LCC protocols are passing around, an active agent might lookup the LCC clause's identically defined rule to trace runtime update, associated with an interaction model being under maintenance by a business expert through a visual configuration tool. Future work includes examining such complementary models and applying AAM to the test-bed of MIAKT and HealthAgents.

Acknowledgements

This work is supported under the OpenKnowledge and HealthAgents STREP projects funded by EU Framework 6 under Grant numbers IST-FP6-027253 and IST-FP6-027213.

References

[1] Foundation for Intelligent Physical Agents, http://www.fipa.org/.

[2] Robertson, D. *et al.*, "Open Knowledge: Semantic Webs Through Peer-to-Peer Interaction", OpenKnowledge Manifesto, 2006, http://www.openk.org/.

[3] Sommerville, I., "Integrated Requirements Engineering: A Tutorial", *IEEE Software* 22(1): 16-23, 2005.

[4] Sommerville, I., "Software Construction by Configuration: Challenges for Software Engineering Research", *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*, p. 9, 2005.

[5] Sommerville, I., "Construction by Configuration: A New Challenge for Software Engineering Education", invited lecture, JENUI'05, Spain, 2005.

[6] Object Management Group, Inc., 250 First Ave. Suite 100, Needham, MA 02494, USA.

[7] Kleppe, A., Warmer, J. & Bast, W., *MDA Explained: The Model Driven Architecture: Practice and Promise*, Addison-Wesley, 2003.

[8] Meservy, T. & Fenstermacher, K., "Transforming Software Development: An MDA Road Map", *IEEE Computer* 38(9): 52-58, 2005.

[9] France, R., Ghosh, S. & Trong, T., "Model-Driven Development Using UML 2.0: Promises and Pitfalls", *IEEE Computer* 39(2):59-66, 2006.

[10] Mellor, S. & Balcer, M., "Executable UML: A Foundation for Model Driven Architecture", *Addison-Wesley*, 2002.

[11] Object Management Group, "OMG Unified Modeling Language Specification (Action Semantics)", *OMG document ptc/02-01-09*, 2002.

[12] Xiao, L. & Greer, D., "The Agent-Rule-Class Framework for Multi-Agent Systems", Special Issue on Agent-Oriented Software Development Methodology, *Multiagent and Grid Systems - An International Journal*, Number 4, Volume 2, IOS Press, to appear.

[13] Robertson, D., "A Lightweight Method for Coordination of Agent Oriented Web Services", *Proceedings of AAAI Spring Symposium on Semantic Web Services*, Stanford, 2004.

[14] Yoder, J.W. & Johnson, R., "The Adaptive Object-Model Architectural Style", *Proceedings of the 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance*, pp. 3-27, 2002.

[15] Fowler, M., "Using Metadata", *IEEE Software* 19(6): 13-17, 2002.

[16] Robertson, D., "A lightweight coordination calculus for agent systems", *LNCS* 3476:183-197, Springer, 2005.

[17] Walton, C., "Model checking multi-agent web services", *Proceedings of AAAI Spring Symposium on Semantic Web Services*, California, USA, 2004.

[18] McLaughlin, B., *Java & XML Data Binding*, O'Reilly, 2002.

[19] JADE platform, http://jade.tilab.com/.

[20] Widener, P., Eisenhauer, G., Schwan, K. & Bustamante, F.E., "Open Metadata Formats: Efficient XML-Based Communication for High Performance Computing", *Cluster Computing* 5(3): 315-324, Springer, 2002.

[21] Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M. & Steggles, P., "Towards a Better Understanding of Context and Context-Awareness", *LNCS* 1707:304-307, Springer, 1999.

[22] Xiao, L. & Greer, D., "A Hierarchical Agent-oriented Knowledge Model for Multi-Agent Systems", *Proceedings of the Eighteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'06)*, pp.651-656, 2006.

[23] Xiao, L., "The Adaptive Agent Model", PhD thesis, Queen's University Belfast, 2006.

[24] HealthAgents project, http://www.healthagents.net/.