Multi-agent Designs for Ambient Systems

Kendall E. Nygard
North Dakota State University
Dept. of Computer Science
Fargo, ND 58105-5164
1-701-231-8203
Kendall.Nygard@ndsu.edu

Dianxiang Xu
North Dakota State University
Dept. of Computer Science
Fargo, ND 58105-5164
1-701-231-8185
Dianxiang.Xu@ndsu.edu

Jonathan Pikalek
North Dakota State University
Dept. of Computer Science
Fargo, ND 58105-5164
1-701-231-8562
Jonathan.Pikalek@ndsu.edu

Martin Lundell
University of Minnesota Crookston
Dept. of Math, Sci., and Technology
Crookston, MN 56716
1-218-281-8180
mlundell@umn.edu

ABSTRACT

Designing and developing software for an ambient intelligence (AmI) system involves difficult challenges related to the varied roles of many heterogeneous devices and communication channels, and intelligent user interfaces. Because ambient systems have unpredictable requirements and are context-aware, software designs must support dynamic and sustainable change. We argue that such designs should utilize formal methods and aspect-oriented techniques, to help in supporting model validation and verification. Features of an aspect-oriented, multi-agent, architectural description language are presented as a mechanism for reasoning about cross-cutting concerns.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Domain-specific Architectures

General Terms

Design, Languages, Theory, Verification.

Keywords

Architecture description languages

1. INTRODUCTION

The vision for systems with ambient intelligence foresees large numbers of processors and tiny sensors integrated into everyday objects, leading to the disappearance of traditional input and output media. In ambient systems there is a high level of shared situational awareness among widely distributed units, structured to promote collaboration, self-synchronization, agile and fast responses to new information, adaptation, and sustainability.

The building of an AmI environment is challenging. Such an environment involves the participation of heterogeneous devices, forming an open, dynamic system, where the available resources, context and activities change continuously. The approach that we take to architecting ambient systems is influenced by our previous work involving distributed systems with humans and agents working together and allocating tasks in applications such as the semi-autonomous cooperative operation of systems of unmanned air vehicles, sensor networks, and sense-and-respond logistics systems [1, 2, 3, 4, 5]. We see a large gap between the abstraction of the high level user needs and the native functionalities provided by the devices. This calls for a flexible design approach in which adaptation to the user is the result of the dynamic building of applications from available resources and the ongoing reconfiguration of applications to adapt to changes in the environment [6]. Multi-Agent Systems (MAS) are a natural and powerful approach to designing AmI systems to function within complex environments.

One powerful capability of intelligent software agents is that they can anticipate events and adapt to changes in their environment. Agents can conduct proactive actions to seek goals and follow their beliefs that pertain to situations that they encounter. They also can actively communicate and collaborate with other agents to achieve objectives that are broader than their own. Due to being massively distributed, software agents for these types of systems must be imbued with decision choices that they can make autonomously, resulting in overall outcomes that are non-deterministic. We follow a formal method approach to multiple agent design for ambient systems. In addition, we identify multiple important cross-cutting software concerns that should be formally included in the architecture. Our work provides a specification approach that employs an architecture description language (ADL) with XML conventions. An example scenario is presented.

In ambient systems there is a clear need for decision support for agents to form teams of agents and assume roles within teams. We present an optimization model that can be employed to provide decision support of this type.

2. MULTI-AGENT SYSTEMS

A software agent is an encapsulated software system situated in an environment where it can conduct flexible and autonomous actions to meet its design objectives [7, 8]. Agent goals can be common or private. The key characteristic of a software agent that is distinctive from other programming paradigms is that within a context, agents persistently evaluate a suite of options that are available to them, then choose among them and act. In contrast, other programming paradigms are much more prescribed.

AMBI-SYS 2008, February 11-13, Quebec, Canada

DOI 10.4108/ICST.AMBISYS2008.2925

In an ambient system, agents are defined for roles such as monitoring the activities and intentions of users, monitoring the status of certain other agents, and devices, brokering the completion of computational tasks among available resources, and representing preferences of users and other agents. Thus, we see a correspondence between the capabilities of a software agent from a programming perspective, and the roles that must be fulfilled in responsive and adaptive ambient intelligence activities. The agents can be homogeneous or heterogeneous, and cooperative or competitive. The agents in an AmI system are heterogeneous and cooperative. Heterogeneity arises from the devices sensing widely differing things, activating diverse controllers, negotiating the completing of varied tasks, and finding acceptable solutions to various models. They are cooperative in that they all contribute to a larger overall objective, even if they have their own localized goals to achieve.

The Belief-Desire-Intention (BDI) agent model can flexibly handle the entire agent modeling in an AmI system. In this model, beliefs correspond to the state of the agent, including the current characteristics of the agent of the environment in which it is functioning. Desires correspond to the effects that the agent attempts to cause in its environment. Intentions represent the plans that the agent has available and is following to realize those effects.

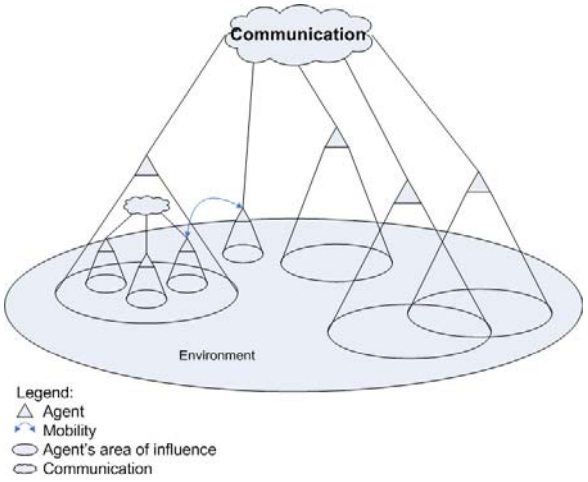

Figure 1. Generic Multi-agent System

Figure 1 generically illustrates a MAS [1]. Some agents are composites formed from subagents, such as those that negotiate, broker, or use the services of other agents. The triangles represent agents or composite agents. The ovals represent the area of influence of an agent, consisting of the entities in an environment to which it can sense and respond. Managing heterogeneous communication networks, alternative and widely varying agent communication protocols, and inter-agent message information content are central issues in the MAS approach. This is illustrated by the network cloud in figure 1. Communication agents often with multiple receiving and transmitting capabilities, specify the work of the communication network itself, and handle issues like whether devices are within listening range of each other and can successfully communicate. Protocol issues, such as the responsibility for logical agents to forward packets, are handled by agents charged with choosing, enforcing and modifying protocols as needed (for example, to support self-healing capabilities). Finally, agents must understand the syntax, parse, process, and act upon the content of messages that support information sharing and cooperative decision making. Some agents could be mobile, having the ability to move among supporting platforms. In an AmI system, mobile agents can model the preferences of brokering agents, users, and the suppliers of resources. These agents can adaptively respond to dynamic conditions, using multiple options available to them to guide them in their quest for carrying out a command or request. Mobility is represented by a double ended arrow, meaning that the agent can move from one environment to another. Mobility is a cross-cutting concern that is not present in most MAS problem domains, but can be powerful or necessary in the ambient system domain.

3. CROSS-CUTTING CONCERNS

The need to manage complexity while creating software that is flexible, adaptable, and evolvable is critical in an AmI system. Although modularity is essential, some requirements and properties cannot be modularized because they are inherently crosscutting concerns. Crosscutting concerns are difficult, because they are inherently distributed, and make other functional components of a software system less self-contained. In a MAS, for example, secure communication among the agents that represent decision-making entities is an important concern regardless of their specific functions. In software engineering it is well-understood that cross-cutting concerns can easily result in expensive duplication of code across many components. Such duplication decreases software quality and makes it difficult to reason about concerns at the architectural level. An application with numerous crosscutting concerns often results in software that is difficult to modularize, understand, reuse, or evolve. In dealing with cross-cutting concerns, it is useful to employ high level abstractions, to make the design of the software intellectually manageable. High-level abstraction often reveals concerns and can provide insight into how to encapsulate them into separate components. It is then possible to precisely describe how the components interact with each other so that the system to behave meets its intended purposes. The MAS manages complexity by separating the concerns into agents, objects, and the environment in which they interact. Some concerns of a system, such as security, resist such modularization.

Aspect-oriented software development (ASOD) addresses the modularization of cross-cutting concerns by separating the cross-cutting concerns into aspect modules. Code that addresses a cross-cutting concern is called advice and is maintained within the aspect. The aspect catalogs the places within the system that need advice and weaves advice into locations called join points. Aspects provide a powerful approach to handling cross-cutting concerns in an AmI system. In an AmI system, concerns are associated with the interaction, adaptation, and autonomy of agents, as well as mobility, learning, and collaboration.

We consider a scenario in which a customer enters a shopping mall with a shopping list stored on her phone. Upon entering the mall, the phone handshakes with the network in the mall network, accessing databases of inventories of stores in the mall. Matching the items on the list with descriptions of products in inventory, the mall network returns a list of products, prices, and store locations in the mall. Based on previous purchases of the customer along with current popular mass market purchases, the phone filters the

returned list, only showing products it anticipates the customer wishes to buy. The customer quickly locates the items she wants and uses credit information stored on her phone to make the desired purchases. Suppose further that upon walking past a music store a video screen displays the cover of a newly released album from one of the artists the customer has mp3 files of on her phone. The shopper stops and makes an unplanned purchase of an album of one of her favorite artists. Finally, her purchasing list is influenced by the interrogated balance available in her credit line, and by the amount of time she has available to shop before she must leave to meet other obligations.

The scenario clearly indicates the need for adaptation in several ways. In this type of scenario, there is a need for personal devices to have the capability of being dynamically configured to work within their environments. Table 1 provides a list of cross-cutting concerns that apply to the agents in the scenario. These concerns are candidates for modularization into aspects, potentially improving the design, testing, and maintenance of the system.

The disparate entities in the scenario have distinct roles and responsibilities that can be represented as individual agents. Each entity has its own set of objectives and goals (modeled as Desires), intelligence (Beliefs) and plans (Intentions) in a MAS. Each unit has a collection of capabilities, sensors and effectors and a limited sphere of influence. Each unit has a measure of local control and enough autonomy to choose a course of action to realize its goals. As the situation evolves, the entities must recognize a change of state, to acquire knowledge, and share it appropriately with others. Collaboration and sharing of capabilities is essential. When changes occur, plans must be adapted to reach their goals. Such adaptation is triggered by knowledge acquisition and may require collaboration. The interactions must be secure and authentication protocols and policies must be enforced.

Although the units are heterogeneous and not all have the same beliefs, goals, intentions, or capabilities, they all are subject to the concerns shown in Table 1. Aspect-oriented development provides a way to reason about all the concerns in individual modules, separate from each of the different agent types.

Table 1. Example Cross-cutting Concerns

Ambient Systems Cross-cutting Concerns	***Definition***	***Application Example***
Autonomy	The ability of agents to act independently.	Customer phone independent of mall network
Communication	Interaction protocols related to message passing, sensors, and effectors.	Protocols and message content concerning sharing information regarding resources, position,
Collaboration, Negotiation, Coordination, Dissolvation, Reliability	The ability to work together or negotiate to achieve common or private goals including the protocols for creating and dissolving partnerships.	Collaboration with mall to retrieve popular mass market purchase data related to customer's list.
Knowledge Acquisition (Learning and Sharing)	The ability to gain knowledge and how to share knowledge with others.	Phone updates buying patterns after purchase
Anticipation, Forecasting	The ability to use past behaviour to predict future needs.	Phone anticipating desirable products from mall search results.
Mobility(a concern for a subset of MASs)	The ability to move from one environment to another.	Ability of phone to "handshake" with mall network and operate within it's domain.
Dynamism, Adaptation, Stability (preconditions and validity of goals)	The ability to react to changes in the environment.	Phone responds to request from music store regarding artists on the phone to the benefit of the customer.
Environment	The environment may place constraints on agents and/or objects and their actions .	Handling connectivity, time issues, power/battery resources, number/frequency/validity of network requests.
Security, Trust, Authentication, Policy enforcement	Protocols for identifying agents, safeguarding knowledge, establishing trust relationships and enforcing policies (e.g., Agent role hierarchies).	Clearances for sensitive information, protocols for establishing contacts and sharing information, and encryption of data.

4. FORMAL METHODS AND AGENTS

The agents in a MAS must exhibit both proactive and reactive behaviors. For example, an agent who is proactively monitoring and predicting the locations and availability of prescribed resources, may need to suspend the monitoring if asked to quickly and reactively allocate resources in response to a perceived change in a user's preferences and plans. Also, some agents must resolve conflicts among multiple objectives. Suspending activities can result in lead to missing critical changes in information that in turn produce actions that may not be required. Yet another issue is that the aggregate behaviors of all the agents may reach beyond the sum of their actions. Thus, in ambient systems, it may be necessary to support controls that limit undesirable emergent effects. Controlling such situations must be done through managing the details of the specification and design of the system. We assert that formal software engineering specification and design practices provide the ability to reason about how and why such problems can occur.

In our MAS, we use an architectural description language (ADL) for identifying the components for agents, objects, and aspects and how they interact with each other. An ADL explicitly describes components and their interfaces, connectors, and architectural configurations [11]. Our approach utilizes multiple software agents with aspects within a framework and notation that resembles Z and, AspectZ [12,13].

Figure 2 conceptualizes the role of the connector in the ADL. The Aspect Component encapsulates the cross-cutting concern but it is the Aspect Connector that contains the information concerning where and how the cross-cutting concern is implemented in other components. In Aspect-oriented programming language, the aspect connector declares a set of join points and weaving rules for the cross-cutting concern. Agent components interface with their environment through sensors and effectors. These sensors and effectors make up the set of possible join points that the aspect connector can target. The links in the ADL represent the configuration of the system, linking the aspect connector to its stated join points in the agents.

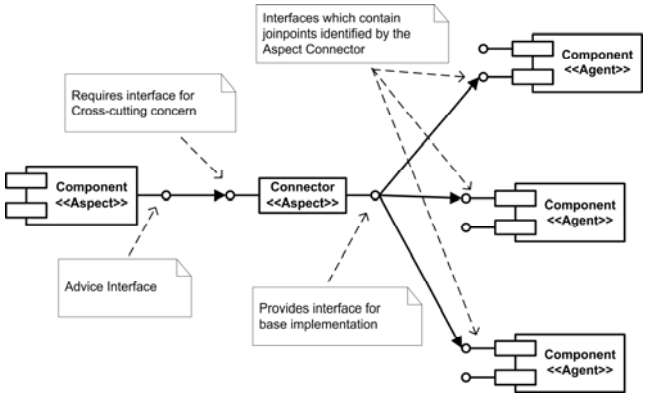

Figure 2. Illustration of an Aspect-oriented MAS ADL

We choose to represent our architecture in the eXtensible Markup Language (XML). In the Figure 3 we display an abbreviated instance of our ADL describing a concern that cuts across two components. The cross-cutting concern is one of recognizing the communication specification of a device. This would be a concern of any agent who needs to establish communication across a wide range of devices that may enter that agent's area of influence. This concern would tend to be updated frequently as newer versions of communication specifications are continually being released and new specifications are regularly introduced.

The two agent components (Figure 3a) realize an interface of type specRec_type (Figure 3d) as sensors. The aspect component, ConfigureComm (Figure 3a), is described to contain functionality related to identifying a communication specification. It is the connector (Figure 3b) that identifies joinpoints as any component that realizes the specRec_type interface. The links (Figure 3c) establish the configuration via the id and href attributes.

(a)

```
<aspectComponent id="ConfigureComm">
  <description>
    Communication specification mechanism
  </description>
  <interface id="IConfigureComm">
    <direction>out</direction>
  </interface>
</aspectComponent>

<agentComponent id="HallAgent">
  <description>...</description>
  <interface id="HA_Interface">
    <effectors>...</effectors>
    <sensors>
      <sensor id="HA_determineCommSpec">
        <type href="#specRecog_type" />
      </sensor>
    ...
    </sensors>
  </interface>
  <beliefs>...</beliefs>
  <desires>...</desires>
  <capabilities>...</capabilities>
</agentComponent>

<agentComponent id="StoreAgent">
  <description>...</description>
  <interface id="SA_Interface">
    <effectors>...</effectors>
    <sensors>
      <sensor id="SA_determineCommSpec">
        <type href="#specRecog_type" />
      </sensor>
      ...
    </sensors>
  </interface>
  ...
</agentComponent>
```

(b)

```
<aspectConnector>
  <crossCuttingInterface id="specificationRecog.in">
      <adviceType>before</adviceType>
  </crossCuttingInterface>

  <baseInterface id="specification Recog.out">
    <pointcut>
      <description>
        Determines communication where agent is
        attempting to determine a device's
        communication specification.
      </description>
      <joinpoint>
        <signature href="#specRecog_type" />
        <kind>execute</kind>
      </joinpoint>
      _
    </pointcut>
  </baseInterface>
</aspectConnector>
```

(c)

```
<link id="link1">
  <description>
    IConfigureComm to specificationRecog.in
  </description>
  <point>
    <anchor href="#IConfigureComm"/>
  </point>
  <point>
    <anchor href="#specificationRecog.in" />
  </point>
</link>

<link id="link2">
  <description>
    specificationRecog.out to HA_determineCommSpec
  </description>
  <point>
    <anchor href="#specificationRecog.out"/>
  </point>
  <point>
    <anchor href="#HA_determineCommSpec" />
  </point>
</link>

<link id="link3">
  <description>
    specificationRecog.out to SA_determineCommSpec
  </description>
  <point>
    <anchor href="#specificationRecog.out"/>
  </point>
  <point>
    <anchor href="#SA_determineCommSpec" />
  </point>
</link>
```

(d)

```
<interfaceType id="specRecog_type">
   <description>
      Determine communication specification of devices
      in range.
   </description>
</interfaceType>
```

Figure 3. ADL Code

The use of XML in ADLs has the clear advantage of its widespread adoption for information interchange and tool support. However, XML is not as precise in its description of properties and behaviors as other notations (e.g., Z), so have supplemented the XML based ADL with more formal notations.

5. DECISION SUPPORT

Task allocation is a critical function in an agent-oriented architecture for an AmI system. From the standpoint of cooperatively making and executing intelligent decision choices, a modeling framework is required. Available modeling frameworks that we have worked with include fuzzy decision rules, neural networks, contract nets, and rough sets. Here we describe a distributed optimization model. We assume that there is a collection of goals to be met, and that the environment in which the agents function to meet the goals is dynamically changing. Goals are met by forming cooperative teams of agents that are collectively capable of meeting the goal at some measurable level of quality. The agents on a given team each carry out a role that contributes to meeting the goal. The aim of the model is to specify optimal decisions concerning which teams are formed and which agents comprise each team. A related model is given in [10]. We adopt the notation given below.

$Goal_t$	Goal to be pursued by team t
$Roles_t$	Number of roles available in team t
$Value_{t,r,a}$	Value of agent a joining team t to carry out role r
$Formed_t$	Binary variable with value 1 if team t is formed
$Teamed_{t,r,a}$	Binary variable with value 1 if agent a joins team t to carry out role r

Teams can have differing numbers of roles, and agents are assignable only to roles for which they can potentially contribute value. Thus, the index sets for the parameters and variables are generally not complete. The optimal assignment of agents to roles and teams is met when the overall utility is as high as possible, which is met by forming teams that objective is to maximize the function below.

$$\sum_{t,r,a} Value_{t,r,a} * Teamed_{t,r,a}$$

Assuming that each goal is to be met at some level by at most one team (to ensure deconfliction among agents), the following condition must be met for each goal g:

$$\sum_{Goal_t = g} Formed_t \leq 1$$

To ensure that all necessary roles are filled by agents on all teams that are formed, the following condition must be met for all teams g and roles r:

$$\sum_{a} Teamed_{t,r,a} = Formed_t$$

Finally, to ensure that all agents have an active role on some team, even if just assigned to staying alert, we form conditions so each agent a must satisfy the following:

$$\sum_{g,r} Teamed_{g,r,a} = 1$$

We stress that parameterization of this model can greatly limit the number of combinations of ways that agents can be assume roles on teams, effectively reducing the otherwise exponentially large numbers of combinations. Also, for a given set of values for the variables $Teamed_{g,r,a}$, the collapsed problem is an easily solvable capacitated transshipment problem with all integer solutions, a powerful property that invites embedding in parent solution procedures. Natural heuristics are also easily formulated. In general, answers to questions concerning how task allocation is best accomplished in AmI systems is in its infancy.

6. CONCLUSION

We developed a formal software engineering approach with an Architectural Description Language (ADL) for specifying multiple interacting software agents for AmI systems. The design

is aspect-oriented and modularizes cross-cutting concerns in a MAS. The use of formal methods has advantages in supporting specification, design, validation, and verification. The design captures the need for dynamic adaptation and non-determinism of AmI systems. XML is employed for the ADL. The use of aspect-oriented concepts, formal methods, and ADLs in a MAS is promising for managing the complexity of the software. This work is an early effort to address the cross-cutting concerns in a MAS ADL. AmI systems are a good fit for the approach, and provide an interesting test base.

7. REFERENCES

[1] Lundell, M., Xu, D., Tolliver, D., and Nygard, K.E. A Multi-agent Design for Sense and Respond Logistics, preprint, computer science, North Dakota State University, submitted for publication, 2007.

[2] Xu, D., Volz, R.A., Miller, M.S., and Plymale, J. Knowledge-Based Human-Agent Teamwork for Distributed Training. *International Journal of Intelligent Control and Systems*. Vol. 11, No. 1, pp. 1-10, March 2006.

[3] X. Du, M. Zhang, K. Nygard, M. Guizani, and H. H. Chen, Self-Healing Sensor Networks with Distributed Decision Making, *International Journal of Sensor Networks*, accepted, to appear.

[4] Lundell, M., Tang, J., Hogan, T., and Nygard, K.E. Agent-oriented Simulation of Cooperative UAV Missions, *WSEAS Transactions on Systems*, 5(4), 2006.

[5] Nygard, K. E., Altenburg, K., Tang, J., Schesvold, D., and Pikalek, J. Alternative Control Methodologies For Patrolling Assets With Unmanned Air Vehicles, Algorithms for Cooperative Control, Oleg Prokopyev, Don Grundel, Robert Murphy, and Panos M. Pardalos, Eds *World Scientific Series on Computers and Operations Research*, Vol. 5, 2007.

[6] Saif, U., Pham, H., Paluska, J.M., Waterman, J., Terman, C., Ward, S.: A case for goal oriented programming semantics. In: *Workshop on System Support for Ubiquitous Computing (UbiSys'03), 5th International Conference on Ubiquitous Computing (UbiComp 2003)*.

[7] Jennings, N., Wooldridge, M., Agent-Oriented Software Engineering, in *proceedings of the 9th European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, 2000.

[8] Kolp, M., Giorgini, P., and Mylopoulos, J., An Organizational Perspective on Multi-agent Arcitectures, in *Proceedings of the 8th Int. Workshop on Agent Theories*, Architectures, and Languages, 2001.

[9] Mathieu Vall´ee1, Fano Ramparany1, and Laurent Vercouter, Dynamic Service Composition in Ambient Intelligence Environments: a Multi-Agent Approach, in *Proceedings of the First European Young Researcher Workshop on Service-Oriented Computing*, UK, 2005.

[10] Hennebry, Michael J., Kamel, Ahmed, and Kendall E. Nygard, An Integer Programmng Model for Asssigning Unmanned Air Vehicles to Tasks, in Sergiy Butenko, Robert Murphey and Panos Pardalos, eds., *Recent Developments in Cooperative Control and Optimization*, Kluwer Academic Publishers, Dordrecht, Netherlands, 2003.

[11] Medvidovec, N., and Taylor, R. A Classification and Comparison Framework for Software Architecture Description Languages, *IEEE Transactions on Software Engineering*, 26(1), pp. 70-93, 2000.

[12] Spivey, J., *The Z Notation: A Reference Manual*. 2nd Ed. Prentice Hall., 1992

[13] Yu, H., Liu, D., Yang, L., He, X. Formal Aspect-Oriented Modeling and Analysis by AspectZ, *Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'05)*, Taipei, 2005.