Synergy between Software Product Line and Intelligent Pervasive Middleware- a PLIPerM Approach

Weishan Zhang
Computer Science Department
University of Aarhus
Aabogade 34, 8200 Aarhus N, Denmark
{zhangws}@daimi.au.dk

Abstract

The current pervasive middleware is designed to have a monolithic structure according to a 'one-size-fits-all' paradigm, which lacks necessary flexibilities for optimization, customization and adaptation to different situations. Ontology evolution is another problem that has not been addressed by the existing pervasive middleware. In this paper, we propose the marriage of software product line and intelligent agent based pervasive middleware in order to alleviate these problems, in the context of an ongoing research project called PLIPerM. Frame based software product line techniques are used together with OWL ontology reasoning enhanced BDI (Belief-Desire-Intention) agents, which are the basic building blocks of PLIPerM. Besides the advantages of a software product line, our approach can handle ontology evolution and keep all related assets in a consistent state. Other advantages include the ability to configure Jadex BDI agents for different purpose and to enhance agent intelligence by adding logic reasoning capabilities indirectly to agent beliefs.

1. Introduction

In the pervasive computing environment, we face the problems of how to tackle the diversity of network technology, and how to handle the diversity of related computing facilities, ranging from powerful computers to small mobile phones and even tiny sensors. As user requirements change and technology develops, it is also becoming more urgent that we can develop and deploy various personalized mobile applications more efficiently and economically. Pervasive middleware is the underlying technology that is supposed to address these challenges, which is attracting a lot of research attentions over the years.

On the one hand, there are quite a lot of pervasive middleware (or mobile middleware as it is used to be) with different flavors, for example event-based, CORBA-based, and most recently OWL [1] ontology based, as has been surveyed in [2]. On the other hand, we agree that the research on pervasive middleware has not taken full advantage of the research results and achievements from software engineering [3], especially software product line. The current pervasive middleware works in a 'one-size-fits-all' manner which is not suitable for the range of hardware/software platforms, networks, and lacks the flexibility for customization and adaptation to different situations as pointed out also in [4]. For example, different type of mobile devices or embedded devices may require specialized optimizations for power consumption, resource utilization, and so on.

As one of the successful research directions in software engineering, software product line research could contribute to the research of pervasive middleware. Commonality and variability management techniques from software product line can be applied to handle domain variabilties for optimizations, customization, and adaptation. This can be achieved by

a pervasive middleware product line, and different members target different platforms and/or different purposes for different application domain.

In this paper, we present our approach to combine the software product line techniques with the intelligent pervasive middleware, in the context of an ongoing research project called PLIPerM (Product Line enabled Intelligent Pervasive Middleware). In PLIPerM, we are applying product line based ontology management and processing mechanisms in order to handle ontology evolution and to keep all affected artifacts in a consistent state. We use enhanced BDI agents [5] as our basic building block for the pervasive middleware, where OWL ontology logic reasoning will add indirect facts to the belief sets of the agents. Initially we are using Frame concepts and techniques [6] as our product line technology, and in most situations we use its XML-based implementation called XVCL (XML based Variant Configuration Language) [7] across both J2SE and J2ME platform.

In XVCL, meta-variables and meta-expressions are the basic means for parameterization. Meta-variables have global scope in a specific branch of a tree formed by meta-components and their values are propagated across the lower meta-components, which can chain together modifications of multiple components (at multiple break points) related to variant features of a considered domain. With the 'same-as-except' and 'composition with adaptation' concept behind the scene, XVCL provides a powerful template mechanism to deal with the adaptation and configuration problems. The <*adapt*> command could include other components adaptively together with <*insert-before*>/<*insert*>/<*insert-after*> and <*break*>. <*select*>/<*option*>/<*otherwise*> and <*ifdef*>/<*ifndef*> provide the control flexibility for options. For details on the philosophy behind Frame and XVCL, and how XVCL works, please refer to [7].

The rest of the paper is structured as follows: in Section 2 we briefly discuss the related pervasive middleware researches and point out some shortcomings with the current pervasive middleware; followed is the section to present an overview of PLIPerM. In Section 4 and Section 5 we demonstrate how to handle ontology evolution with the product line based approach using examples of instance changes and concept changes. BDI agent configurations based on product line techniques are illustrated with examples in Section 6. Conclusions and future work end the paper.

2. Related work

There exist some mobile-agent-based pervasive middleware, such as SOMA (Secure and Open Mobile Agent) [8], ACAI (Agent-based Context-Aware Infrastructure) [9] and SOCAM (Service-oriented Context-Aware Middleware) [10]. SOMA uses CORBA to interact with other mobile objects relying on a central server. CORBA is based on a client-server model of interaction which is not well suited to the more peer-to-peer world in pervasive computing. ACAI is based on the ontology context model, but there is no service adaptation mechanism, and user-centered contexts are not taken into consideration in its context models. SOCAM has a distributed structure but the context is stored centrally. Its implementation is based on RMI which is not available for J2ME/CLDC-enabled mobile devices. None of these middleware is utilizing the practical reasoning mechanism provided by BDI agents, and software product line techniques neither.

As stated in [4], it is not possible to build a middleware fulfilling all the requirements of different applications and being economical in terms of resource consumption at the same time. Therefore an Aspect Oriented Programming (AOP) based product line approach was proposed to build application specific middleware.

Dynamic aspect weaver was used to realize runtime configuration [4] where the idea of a product family is used to configure the middleware for specific purpose. According to our experience and also as mentioned in [4], dynamic AOP will result in considerable performance overhead. In our opinion, AOP itself is not a very good product line technology as explored in our former work [11]. At present stage, dynamic AOP will be too heavy weight to be suitable to pervasive computing.

Similar ideas to combine the software engineering and distributed computing principles were proposed in [3], where Mixin layer approach was explored as a ubiquitous middleware product line implementation technology. Configurability, reusability and extensibility of ubiquitous middleware could benefit from the software product line architecture. In the mean time, it should be noted that it is not always possible to implement a feature as a single mixin layer. Such kind of product line technology in programming language level only is not sufficient enough to handle different kind assets for a middleware product line, e.g. OWL ontologies used for context modeling. It is really hard for mixin to handle those high level variabilites, such as necessary configuration for different mobile agent platforms when software agent is applied to build pervasive middleware.

MobCon [12] can also be considered as a product line based middleware. It can handle variability in technical concerns such as data persistence, screen management, and session management with a container abstraction. However, there is no mention of how to handle other kinds of variants, and it is hard to trace the variant effects across software assets. MobCon is restricted in its application with MIDP [13]. As for the intelligence, MobCon is really weak with respect to the needed intelligence for personalized service in user-centered pervasive computing.

Take into the surveyed results in [2], a number of problems with the current pervasive middleware can be identified:

- Monolithic structure using the ‘one size fits all’ paradigm is not suitable for the range of hardware, software platforms, available services, networks, etc. in the pervasive computing environment, where specialized optimization and customization may be required for specific situations.
- The existing ontology-based pervasive middleware, did not address how to handle ontology evolution [14], which results from the timely adaptation of an ontology to changes and the consistent propagation of these changes to dependent artifacts. And at the same time, the ontology reasoning capabilities are not fully utilized by the existing pervasive middleware.
- Weak intelligence. The existing agent-based pervasive middleware is using agent for code mobility mainly, not fully exploring the BDI agent capabilities, and not much intelligence is built into the used agents. To make the pervasive middleware practical in user-centric scenarios, practical intelligence is required for the supporting of various decisions.

3. An overview of PLIPerM

3.1 Application of software product line to pervasive middleware

In the context of PLIPerM, software product line can provide the following benefits:

- Helping in the management and aggregation of context ontologies, and further more to facilitate the handling of ontology evolution.

- Helping in the management of PLIPerM components for changes, and hence to facilitate the handling of middleware component evolution.
- Providing opportunities and capabilities for specialized optimizations, customization and adaptation to different situations and domains.

The idea in PLIPerM is as follows: starting from the context modeling, we manage the underlying context knowledge using the techniques of how commonalities and variabilities are handled in a product line, an approach whose feasibility is demonstrated in the previous work [15]. Using this knowledge we build a pervasive middleware product line, in which specialized optimizations and designs could be dedicated to different properties of the underlying hardware and software platforms, networks, contexts etc. in the way shown in [16]. All of these will be implemented based on the service oriented architecture (SOA), as the system should be able to connect with outside service providers. Customizations and configurations will be conducted as early as possible before the middleware executes, as run-time configuration usually involves higher overhead. This is the main reason for us to choose a construction time configuration technology such as XVCL.

There are several other reasons for choosing XVCL as the underlying software product line technology.

- Frame based ontology management and aggregation mechanism can run across both J2SE and J2ME, which makes it suitable for the computing capabilities in pervasive computing environment.
- XVCL has highly flexible template capabilities acting as a meta-ontology management and aggregation mechanism, which makes XVCL an ideal solution for handling ontology evolutions.
- Ontology evolution is far more than the management of ontology itself, changes to other artifacts must be handled in a unified way in order to keep all artifacts in synchronization. Frame based mechanisms could help to achieve these goals.

3.1 Enhancing the intelligence of a BDI agent

PLIPerM focuses on the user-centered application scenarios where a user plays a central role and intelligence is vital for the usability of services. BDI agent is chosen as a basic component in PLIPerM based on the Jadex [17] platform. But Jadex BDI agent itself is not enough to provide with necessary intelligence to support the user-centric idea in PLIPerM. Some of the context information, such as the hobby sharing information, is not encoded in agent beliefs as usually it is not known in advance.

On the other hand, theoretical reasoning can be very useful for reasoning about classification and can be used to derive new information that is not superficial. So the integration of OWL reasoning to provide implicit facts for BDI agents has big potential to make smarter decisions and provide better service. Therefore in PLIPerM, an OWL ontology reasoning enhanced BDI agent is proposed where details are shown in [20]. Context ontologies reasoning results can be used to extend the agent beliefs which do not exist or cannot easily be obtained without the ontology reasoning.

In PLIPerM, the existing reasoning capabilities of OWL will be utilized. This reasoning process could be simplified as the following formula, $CQ=R(O)$ where CQ stands for the results of classification and query used during reasoning, R stands for the actual process of Tbox and Abox reasoning, O stands for the OWL ontology sets used for the current reasoning. After OWL reasoning, there may involves the changes of agent state, e.g. new

beliefs obtained. Therefore, according to [21], a new belief update action α_{ub} : α_{ub} =(*UpdateBelief,* B',B,β, $\triangle t$) can be defined, the action type is *UpdateBelief*, β is the parameter for belief, $\triangle t$ is the interval for updating, B is the original belief, $B'=F(CQ)$ is new beliefs after formatting the reasoning results *CQ*. Therefore it is theoretically feasible to combine the ontology reasoning with BDI reasoning in the context of Jadex platform.

The way to combine the power of practical reasoning and theoretical reasoning in PLIPerM is shown in Figure 1 (the lower part). RacerPro [22] is used as the main reasoning tool because of its maturity and special features for context-aware computing, for example the built-in capabilities of reasoning on region relationships. Ontology reasoning is conducted with a RacerPro server via HTTP protocol.

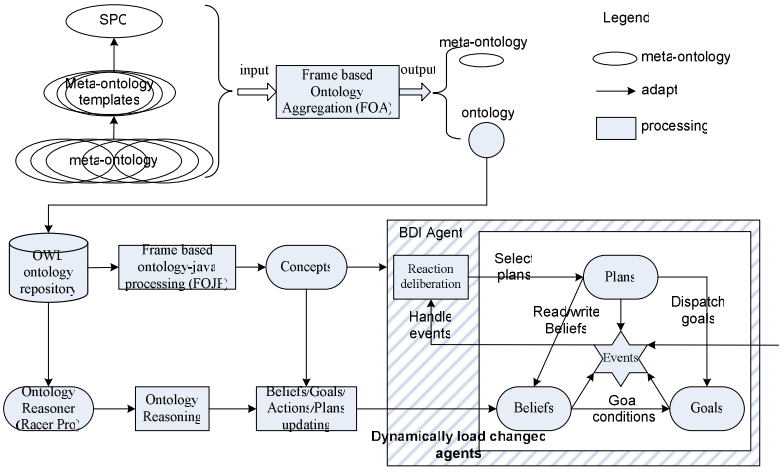

Figure 1. Working flows of PLIPerM

The main components in PLIPerM are Frame based Ontology-Java Processing (FOJP) components. The main functionalities include:

- Bridging the world of OWL ontology and java software agents. It uses the advanced template mechanism of Frame technology to map the concepts and properties in OWL to the related Java class and attributes.
- Management of ontologies and handling of ontology evolution. This will help to keep all ontologies in consistent state.
- Managing the update of agent definitions, including the agent belief, goals, actions and plans. The corresponding java class implementing agent plans may also need to be updated.

In the followed sections, we will demonstrate how to use software product line techniques to handle ontology evolutions, and also to configure PLIPerM agents for different purposes.

4. Product line based ontology evolution handling in PLIPerM

A complete ontology contains every piece of knowledge to be used for all application scenarios, which represents all variants in the domain. It can be huge (for example a Food ontology used in health care scenarios, can reach the size of 3M bytes itself) and keeps evolving. In reality, not all devices will have space to store all the information in the ontology, and not all applications necessarily need all the information. Only parts of the ontologies may be used for different application scenarios and/or different devices.

Therefore an appropriate subset of a complete ontology for a specific situation is enough for resource-constrained devices in pervasive computing environment.

The basic idea applied in PLIPerM is to apply the software product line idea to separate the ontology into parts that usually remains stable, and other parts that change more often. That is to say, a smaller granularity of usable part of a ontology than a complete ontology is used, which is called a meta-ontology, as the basic elements for managing ontologies. Take the person ontology as an example, the background profile is relatively stable and is considered as one meta-ontology for management; but the mood usually changes more often as a person can be in different moods for different events. Similarly, the research profile is a variant part as publications built up and research projects come and go. These different fragments of a complete ontology are the starting points for the creation of meta-ontologies.

In PLIPerM, a meta-ontology is defined as a fragment of OWL code instrumented with XVCL commands, and we use the XVCL namespace <*xvcl:*> to differentiate XVCL commands from other XML tags. Different meta-ontologies can be distributed on different site according to the device resources, quality of service, application scenarios, and so on. When needed, especially when conduct ontology reasoning, different parts of an ontology (in the form of meta-ontology) are synthesized to form a complete one, or partially complete one. This could be conducted via HTTP protocol using Frame based aggregation components, which is available both on J2SE and J2ME platform. On J2ME, KXML [23] based Frame processing components are used.

Based on the product line ideas, we analyzed the developed ontology in order to obtain the common part that is relatively stable and variant part that is usually changing. The structure of ontologies used in PLIPerM showing stable and variant parts is depicted in Figure 2.

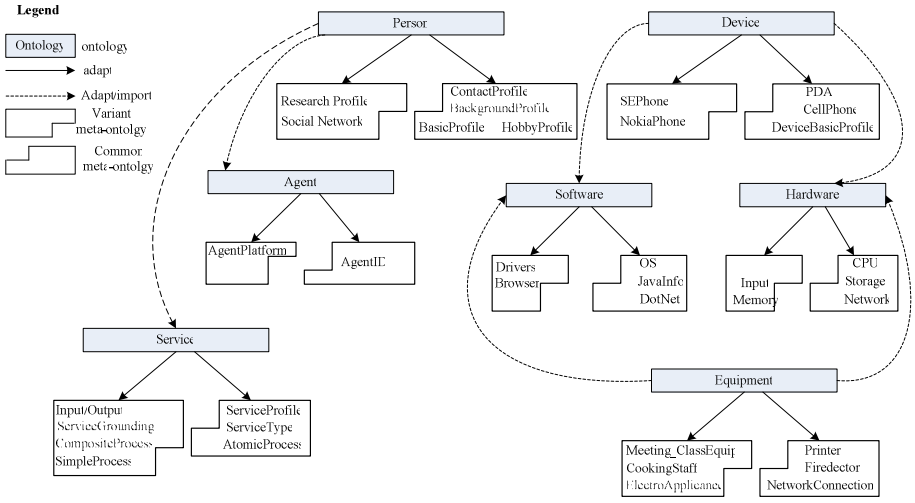

Figure 2. Ontologies structure showing stable and variant parts

Ontology evolutions can affect not only ontology itself, but also the dependent artefacts. The product line based ontology evolution handling process in PLIPerM is shown in Figure 3. To make the handling of ontology evolution easier, all artefacts in PLIPerM are in the meta-level as shown later.

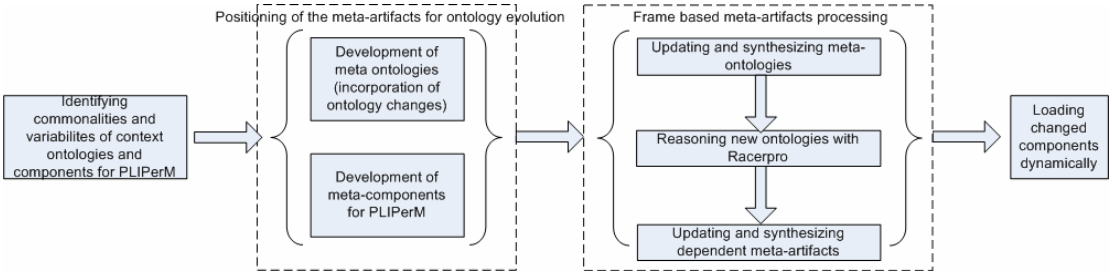

Figure 3. Product line based ontology evolution handling in PLIPerM

After the development of context ontologies and other components for PLIPerM, we then position all these artifacts by adding variation points for the incorporation of possible ontology changes. The positioning for ontology evolution process involves two parallel tasks, namely development of meta-ontologies and development of other meta-artifacts for PLIPerM including code components. For every task, there involves the adding customization slot (variation point) into the meta-artifact using XVCL commands. In reality this process is incremental and iterative in nature in order to make the meta-artifacts more adaptable for ontology evolution.

Then we can utilize the Frame based ontology aggregation and processing components (FOA and FOJP as show in Figure 1) to process the related meta-artifacts when there are ontology evolutions. There are three steps involved in this process. First, the related meta-ontologies will be updated and synthesized based on the specification of needed changes for resolving the ontology evolution by the FOA components; then ontology reasoning tool Racerpro will be invoked to conduct related reasoning in order to get new or updated information from the ontology changes; after that all other dependent meta-artifacts will be updated and synthesized accordingly in response to the ontology changes and also the new or updated reasoning results, i.e. FOJP will propagate these changes to the corresponding Java classes, agent definition files, etc.

At this stage, all artifacts including meta-artifacts are up-to-date. The new or updated components are then parsed if needed, and now they are ready to be loaded dynamically with the dynamic model provided by Jadex[17] [24].

5. Examples of ontology evolution handling in PLIPerM

In PLIPerM, we manage all artefacts in the meta-level, i. e. meta-artefact is the basic building block, in order to facilitate the management of ontology evolution and other evolution changes. This is supported by XVCL working on any textual artefact. For example, an ADF (Agent Definition File) can be incorporated with XVCL customization slots.

Figure 4 is an ADF for a person agent named ‘Weishan’. Beliefs, goals and plans are first class objects in a JadeX agent, which are modeled with XML tags. Java code is used for the implementation of agent actions for the Plan.

```
<agent xmlns="http://jadex.sourceforge.net/jadex"  name="Weishan_agent" package="plimm">
    ……
<beliefs>
    <beliefset name="busyTime" class="String">
      <fact>"Nov 6, 2006 3:00 pm to Nov 6, 2006 5:00 pm"</fact>
    </beliefset>
```

```
    <beliefset name="hobbySports" class="String"><fact> "tableTennis"</fact></beliefset>
    <beliefset name="hobbySportsSharing" class="String"><fact>Soe Maya</fact></beliefset>
<goals>
    <achievegoal name="proposeAppointment" recur="true" recurdelay="1">
      <parameter name="appointment" class="Appointment">
        <bindingoptions>$beliefbase.initial_appointments</bindingoptions></parameter>
      <creationcondition>  $beliefbase.initial_appointments!=null</creationcondition>
      <targetcondition>Appointment.DONE.equals($goal.appointment.getState())</targetcondition>
    </achievegoal>
     ……
</goals>
<plans>
 <plan name="proposeAppointmentPlan">  …... <body>new MakeAppointmentProposalPlan()</body>
 </plan>
    ……
</plans>
</agent>
```

Figure 4. BDI Agent definition of a person

Customization slots are added to the agent definition in order to accommodate evolutionary customization as shown in Figure 5. New beliefs, goals and plans could be inserted to the agent definition into the corresponding *<break>* points using *<insert>/<insert-before>/<insert-after>*. This agent definition will be further refined later.

```
<agent xmlns="http://jadex.sourceforge.net/jadex"  name="Weishan_agent" package="plimm">
<beliefs>
  <xvcl:break name=”busyTime”>
    <beliefset name="busyTime" class="String">
      <fact>"Nov 6, 2006 3:00 pm to Nov 6, 2006 5:00 pm"</fact>
      <fact>"Nov 8, 2006 1:00 pm to Nov 8, 2006 5:00 pm"</fact>
    </beliefset>
  </xvcl:break>
  <xvcl:break name=”newBelief”>  </xvcl:break>
  ……
  <xvcl:break name=” hobbySports”>
    <beliefset name="hobbySports" class="String"><fact> "tableTennis"</fact></beliefset>
  </xvcl:break>
  <xvcl:break name=”hobbySportSharing”>
    <beliefset name="hobbySportsSharing" class="String"><fact>Soe Maya</fact></beliefset>
  </xvcl:break>
<goals>
  <xvcl:break name=”newGoal”>  </xvcl:break>
    <achievegoal name="proposeAppointment" recur="true" recurdelay="1"> ……</achievegoal>
     ……
</goals>
<plans>
  <xvcl:break name=”newPlan”>  </xvcl:break>
 <plan name="proposeAppointmentPlan">  ……<body>new MakeAppointmentProposalPlan()</body>
 </plan>
    ……
</plans>
</agent>
```

Figure 5. Meta-ADF for a BDI Agent

5.1 Examples of instance changes

As an example, Weishan attended a very interesting presentation during a conference, sharing some ideas very relevant to his research project. He had a fruitful talk with the author named Paul during the coffee break and they decided to collaborate with each other on Weishan's project. Now Weishan and Paul both would update their information on their

collaborators in the corresponding meta-ontologies (named Weishan.xvcl and Paul.xvcl, built from Person ontology). This is shown in Figure 6. At the same time, their own person agent definition will be updated as well if there is collaborator information in ADFs.

```
< —Person meta-ontology for Weishan, file name is Weishan xvcl-->
<Person rdf ID= 'Weishan'>
 <givenName rdf datatype="&xsd;string'>Weishan</givenName>
 ……
 <xvcl break name= "collaborators">
    <collaborateWith rdf resource="#Kunz"/>
  ……
 </xvcl break>
```

```
<xvcl adapt x-frame= "Weishan xvcl">
  <xvcl insert-before break=" collaborators">
<collaborateWith rdf resource= "#Paul"/>
</xvcl insert-before>
</xvcl adapt>
……
<xvcl adapt x-frame= "Paul xvcl">
  <xvcl insert-before break= "collaborators">
<collaborateWith rdf resource= "#Weishan"/>
</xvcl insert-before>
</xvcl adapt>
……
```

Figure 6. Handling instance change-case 1

Suppose 'Weishan' changed his address and moved to another city, then this would trigger a Jadex maintenance goal to re-evaluate the sharing hobby information using RacerPro. The returned reasoning results will be parsed and used to update the agent definition for 'Weishan' as shown in Figure 7. The new facts will be propagated and added to the belief base in all the related ADFs who share the same hobby in a similar way.

```
<xvcl:adapt x-frame="Weishan_agent.xvcl">
  <xvcl:insert break="hobbySportSharing">
    <!--parsed reasoning hobby sharing results here -->
  </xvcl:insert>
</xvcl:adapt>
<xvcl:adapt x-frame="Soe_agent.xvcl">
……
</xvcl:adapt>
……
```

Figure 7. Handling instance changes-case 2

5.3 Examples of concept changes

Ontology evolutions involving concept changes will definitely affect ontology itself, and may affect any part of goals, plans and beliefs of an agent. For example, suppose the Person ontology did not model 'Mood' information and we later found that it is essential. Then the Person ontology has to be updated and the person agent including beliefs, plans, goals and actions (for example to show what kind music a person would like to listen to with respect to the mood he is in).

The updating of agent definitions is shown in the left part of Figure 8. We update the agent definition by <*xvcl:insert*>ing the related mood and music beliefs, song playing goal and plan into the corresponding <*break*> points. At the same time, the Person ontology should be updated. To achieve this, we would also use <*xvcl:adapt*>/<*xvcl:insert*> to insert the 'mood' concept class and related properties, instances to the person meta-ontology as shown in the upper right part of Figure 8. The java class implementing the playing song actions (shown in the lower right part of Figure 8) are used and parsed. The changed agent definition and new java class could be loaded dynamically with Jadex dynamic model.

```
<xvcl:adapt x-frame=”Weishan_agent.xvcl”>
 <xvcl:insert break=”newBelief”>
  <belief name="mood"  class="String">
  <fact>"happy"</fact>
  </belief>
  <beliefset name="musicList" class="Music">
  </beliefset>
 </xvcl:insert>
 <xvcl:insert break=”newgoal”>
   <performgoal name="play_song">
    <parameter class="String" name="s_name"/>
   </performgoal>
 </xvcl:insert>
 <xvcl:insert break=”newplan”>
   <plan name="play">
    <body>new PlaySongPlan()</body>
     <trigger> <goal ref="playSong"/></trigger>
   </plan>
 </xvcl:insert>
</xvcl:adapt>
……
```

```
<!--update person meta-ontology -->
<xvcl:adapt x-frame=”person_ontology.xvcl”>
  <xvcl:insert break=”new_concept”>
     <owl:Class rdf:ID="Mood"/>
     <owl:ObjectProperty rdf:ID="inMood">
           <rdfs:domain rdf:resource="#Person"/>
          <rdfs:range rdf:resource="#Mood"/>
     </owl:ObjectProperty>
     ……
  </xvcl:insert>
   </xvcl:adapt>
```

```
<!--play song according to the mood -->
public class PlaySongPlan extends jadex.runtime.Plan {
   public PlaySongPlan() {…}
   public void body () { play();  }
   public void close() {… }
   public void play() {…}
  …..
}
```

Figure 8. Handling ontology evolution of adding concepts

5.4 Discussion

The instance changes including property value changes are the main kinds of ontology evolution. For the existing ontology based middleware like COBRA and ACAI, instance changes usually will not affect those mapped java classes from ontologies. But concept changes will make them cannot function well or not usable at all as they could not be adapted easily to the needed changes, and the dependent artifacts could not be kept in a consistent state. We used the Frame based mapping mechanism which could keep the related artefacts in a consistent state.

According to Noy and Klein [14], there are 22 kinds of changes to an ontology during its evolution. The above meta-artefact based ontology evolution handling mechanism by applying the Frame enabled product line techniques could handle the identified ontology evolution in principle the same as shown in our examples.

Researches on ontology evolution (e.g. [18][19]) mainly focus on keeping semantic consistent and developing comprehensive tool support. Some are also concentrated on ontology versioning. The product line based ontology evolution handling approach in PLIPerM is simple and practical, with the main focus on keeping not only ontologies themselves, but also all other artifacts evolving in a consistent manner, which can be used both in J2SE and J2ME environment. The Frame based configuration mechanisms make the ontology versioning unnecessary in PLIPerM.

6. Product line based configuration for enhanced BDI agents

For software agents, there are a number of variants and commonalities. For example, an agent may or may not use BDI model; agents may use certain communication protocol for example FIPA Contract-Net; agent can use different content language for example OWL; and agent can be implemented with certain platform for example J2SE/J2ME, using different framework for example Jadex or JADE which are open source, or commercial ones for example JACK. This situation is illustrated in Figure 9 as a feature model [25].

In the prototype implementation, we consider agents that negotiating appointments. The appointment can be generic, for example a sport appointment or a social event appointment.

For interaction protocol, FIPA Contract-Net is chosen, and the action plans can be either mobile plans or standard plans. Dynamic models are used in order to load updated agents in case there are changes in agent beliefs, goals and plans.

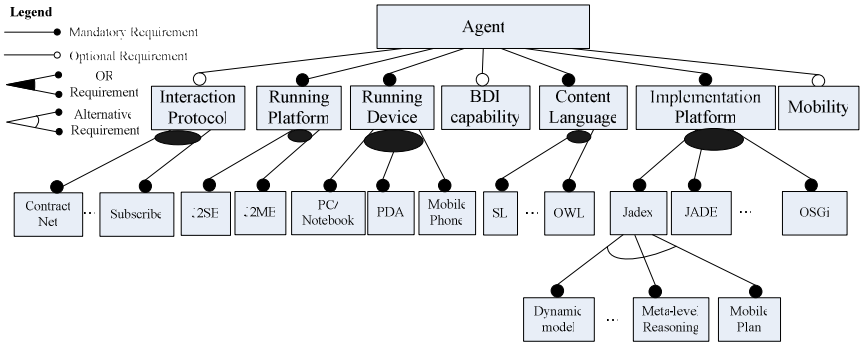

Figure 9. Feature model of agent configurations in pervasive middleware

Now we define a generic agent capability meta-ADF that can be used for both mobile plan and standard plan as shown in Figure 10. If the meta-variable '*mobile*' is defined, then the mobile plan is the option, or else the standard plan will be used.

```
<xvcl:x-frame name="GenericPlan">
  <capabilities>
  <xvcl:ifdef var="mobile"/>
   <capability name="procap" file="jadex.planlib.ProtocolsMobile"/>
   <capability name="dfcap" file="jadex.planlib.DFMobile"/>
  </xvcl:ifdef>
   <capability name="procap" file="jadex.planlib.Protocols"/>
   <capability name="dfcap" file="jadex.planlib.DF"/>
  </capabilities>
</xvcl:x-frame>
```

Figure 10. Generic capability meta-ADF

Then the generic capability meta-ADF can be used to define an agent. As show in Figure 11 an agent is defined to use the mobile plan with <*xvcl:adapt*>ing GenericPlan.xvcl (Figure 10). If standard plan needed, then simply removing the '*mobile*' meta-variable definition will be the solution (i.e. deleting the second line in Figure 11). When compared to Figure 5, the agent definition is spitted into multiple small new meta-ADFs, which are more flexible and adaptable.

```
<agent name="Weishan_agent" package="plimm">
<xvcl:set mobile="Mobile"/>
<xvcl:adapt x-frame="GenericPlan.xvcl"/>
<beliefs>
  <xvcl:break name="busyTime">
    <beliefset name="busyTime" class="String">
      <fact>"Nov 6, 2006 3:00 pm to Nov 6, 2006 5:00 pm"</fact>
    </beliefset>
  </xvcl:break>
  <xvcl:break name="newBelief"> </xvcl:break>
  ……
<goals> …… </goals>
<plans>
  <xvcl:break name="newPlan"> </xvcl:break>
  <plan name="proposeAppointmentPlan">……<body>new MakeAppointmentProposalPlan()</body>……</plans>
</agent>
```

Figure 11. Example agent using mobile plan capability

In Jadex, the APIs for both mobile plan and standard plan are very similar. The code of standard plan is placed in the *body()* method, while the code of mobile plan goes into the *action(IEvent)* method. As FIPA Contract-net protocol is used for the negotiation of appointments, the control flow is the same for both mobile plan and standard plan. Taking all this into consideration, a generic making appointment proposal plan is implemented as shown in Figure 12.

```
<xvcl:x-frame name="MakeAppointmentProposalPlan ">
 <xvcl:ifdef var="mobile"/>
  package plimm.mobile;
  import jadex.runtime.IEvent;
  import jadex.runtime.MobilePlan;
  public class MakeAppointmentProposalPlan extendsMobilePlan
  {
    public void action(IEvent event)
 </xvcl:ifdef>
 <xvcl:ifndef var="mobile"/>
  package plimm;
  import jadex.runtime.Plan;
  public class MakeAppointmentProposalPlan extends Plan
  {
    public void body()
 </xvcl:ifndef>
   { Appointment[] suitableAppointment =
       (Appointment[])getParameterSet("appointment").getValues();
    if(suitableAppointment.length > 0)
    {….}
    }}
</xvcl:x-frame>
```

Figure 12. Generic making appointment proposal plan

Again, the '*mobile*' meta-variable serves as a control flag for standard plan and mobile plan. In this way, a fine granularity of configuration of agents, and also other textual artifacts, can be achieved for different customizations and situations.

Please note that in this paper, no specialized optimizations and customizations are discussed as they are our future work. In principle, they can be treated as variants on the meta-artefacts, which can be handled in the way as how ontology handled and used in this paper. Currently the runtime dynamic adaption and configuration depend on the capabilities provided by Jadex dynamic model and possibly from OSGi framework [26].

7. Conclusions and future work

Pervasive computing system is becoming complex, and pervasive middleware requires a better paradigm and architecture techniques. The current pervasive middleware has a monolithic structure and works in a 'one-size-fits-all' manner which lacks the flexibility for customization and adaptation to different situations. The combination of product line technology with pervasive middleware, especially the combination with the intelligent pervasive middleware, is not yet fully explored, and pervasive middleware could benefit from product line technology.

In PLIPerM, Frame technology (XVCL especially) is used as the supporting product line technology. An enhanced intelligent agent is the basic building block of the pervasive middleware running on a service oriented environment. A meta-ontology based ontology management and aggregation approach is applied to facilitate the usage of ontology. With the support of XVCL, the Frame based product line techniques can help to aggregate all the related context ontologies, and are used to handle ontology

evolution, propagate the required changes to all the affected artifacts, e.g. agent beliefs/plans/goals, which are managed also at the meta level in order to cope with possible evolutions, and facilitating specialized customization, optimization and adaptation for different situation and domains. Then these changed intelligent agents could be loaded dynamically to provide adaptation for different purpose. In this respect, we rely on the capability provided by Jadex at present stage.

To make the user-centered pervasive computing a reality, the supporting infrastructure must be intelligent enough in order to timely, smartly and accurately react to context changes and prepare services accordingly. In PLIPerM, the power of BDI practical reasoning and OWL ontologies theoretical reasoning are combined in order to improve the intelligence of mobile middleware. In this approach, belief sets are enriched with the results of OWL ontology reasoning, where some beliefs cannot be obtained directly or explicitly. In this way, we enhanced the Jadex agent by adding logic reasoning capabilities indirectly to its beliefs.

We are continuing the implementation of the enhanced BDI agents, and especially those agents based on the J2ME version of Jadex. At the same time, the enhanced agents based on OSGi technology are planned, which could make the integration with outside service provider easier. The possibility of using Distributed Description Logics (DDL) [27] is investigated as DDL seems to be closer to the distributed meta-ontologies used in our approach. The performance measurements and optimizations for different situations will be reported when the first full version of the product line is ready.

8. Acknowledgement

The research reported in this paper has been supported by the Hydra EU project (IST-2005-034891).
We thank Lars Braubach and Alexander Pokahr from University of Hamburg for the support of Jadex.
We also thank Racer Systems GmbH & Co. KG for the education license for Racer Pro.

References

[1] Web Ontology Language (OWL) homepage. http://www.w3.org/2004/OWL/. 2007-4-23

[2] W. Zhang, T. Kunz and K. M. Hansen. Product Line Enabled Intelligent Mobile Middleware. Proc. of the Twelfth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2007). Auckland, N.Z. July, 2007. pp. 148-157

[3] S. Apel and K. Böhm. Towards the Development of Ubiquitous Middleware Product Lines. Software Engineering and Middleware: 4th International Workshop, SEM 2004, Linz, Austria, Sep, 2004.

[4] W. Gilani, N. H. Naqvi, O. Spinczyk. On adaptable middleware product lines. Proceedings of the 3rd workshop on Adaptive and reflective middleware, Toronto. pp. 207-213, Oct. 2004.

[5] Definition of BDI software agent. http://en.wikipedia.org/wiki/BDI_software_agent. 2007-6-1

[6] P.G. Bassett. Framing Software Reuse. Yourdon Press, Prentice-Hall, 1996

[7] XVCL method and tool. http://xvcl.comp.nus.edu.sg. 2007-6-2

[8] P. Bellavista, A. Corradi, C. Stefanelli. Mobile Agent Middleware to Support Mobile Computing IEEE Computer, Vol. 34, No. 3, pages 73-81, March 2001.

[9] M. Khedr, A. Karmouch ACAI: Agent-Based Context-aware Infrastructure for Spontaneous Applications. Journal of Network & Computer Applications, Volume 28, Issue 1, pp. 19-44. 2005

[10] T. Gu, H. K. Pung, D. Q. Zhang. A Service-Oriented Middleware for Building Context-Aware Services. Elsevier Journal of Network and Computer Applications (JNCA), Vol. 28, Issue 1, pp. 1-18, January 2005

[11] N. Loughran, A. Rashid, W. Zhang and S. Jarzabek. Supporting Product Line Evolution with Framed Aspects. Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS04), Lancaster, UK. Mar. 2004 PP. 15-19

[12] V. Cepa. Product-Line Development for Mobile Device Applications with Attribute Supported Containers. Ph.D. Dissertation, Darmstadt University of Technology, 2005

[13] Mobile Information Device Profile (MIDP). http://java.sun.com/products/midp/. 2008-1-20

[14] Noy, N.F., Klein, M. Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems No.6, 2004 pp.428-440

[15] W. Zhang and T. Kunz. Product line based ontology reuse in context-aware e-business environment. Proc. of ICEBE 2006, Shanghai. Oct. 2006, pp. 138-145

[16] W. Zhang, S. Jarzabek. Reuse without Compromising Performance: Industrial Experience from RPG Software Product Line for Mobile Devices. Proc. of SPLC2005, Rennes, France, Sept. 2005, LNCS 3714, pp. 57-69

[17] Jadex homepage. http://vsis-www.informatik.uni-hamburg.de/projects/jadex/. 2007-6-2

[18] N. F. Noy, A. Chugh, W. Liu, M. A. Musen. A Framework for Ontology Evolution in Collaborative Environments. 5th International Semantic Web Conference, Athens, GA, 2006.

[19] G. Flouris, D. Plexousakis, G. Antoniou. Evolving Ontology Evolution. Proceedings of the 32nd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 06), Jan. 2006, Merin, Czech Republic

[20] W. Zhang and T. Kunz. Enhancing the Intelligence of Mobile Middleware Environment. Proceedings of the International Workshop on Context-Awareness for Self-Managing Systems. Toronto, May 2007, pp.46-61

[21] A. Pokahr, L. Braubach, W. Lamersdorf. *A Flexible BDI Architecture Supporting Extensibility*, The 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2005).

[22] RacerPro. http://www.racer-systems.com/. 2007-2-3

[23] KXML homepage. http://kxml.sourceforge.net/kxml2/. 2008-1-16

[24] JADE homepage. http://jade.tilab.com/. 2007-2-31

[25] K. Czarnecki and U. Eisenecker. Generative Programming: Methods, Tools, and Applications, Addison-Wesley, 2000

[26] OSGi homepage. http://www.osgi.org. 2007-6-1

[27] L.Serafini and A.Tamilin. DRAGO: Distributed reasoning architecture for the semantic web. In Proc. of the Second European Semantic Web Conference (ESWC'05), Heraklion, Greece.

Authors

Weishan Zhang is an Associate Professor at the Department of Computer Science, Aarhus University, Denmark. His current research is part of the EU HYDRA project, collaborating with Prof. Klaus Marius Hansen. His research interests include intelligent pervasive middleware, software reuse including software architecture and software product line, software engineering and knowledge engineering with embedded system and mobile computing. He visited Department of Systems and Computer Engineering, Carleton University, Canada from Jan. 2006 to Jan. 2007. From Sept. 2001 to Sept. 2003, he was doing post-doctoral research at the Department of Computer Science, National University of Singapore.s