Aspect-Oriented Context Modeling for Embedded Systems

Tomoji Kishi
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsunokuchi
Ishikawa 923-1292, Japan
tkishi@jaist.ac.jp

Natsuko Noda
NEC Corporation
2-11-5 Shibaura, Minato-ku
Tokyo 108-8557, Japan
n-noda@cw.jp.nec.com

Abstract

One of the characteristics of embedded systems is that they depend on their contexts, i.e. they react to the context changes, and their behaviors are constrained by the contexts. In modeling such context-dependent systems, we have the following difficulties: 1) As we have to treat various kinds of contexts such as logical contexts and physical contexts, it is difficult to model them from one point of view. 2) By its nature, the model for internal processing tends to depend on the model of external contexts, and this makes modifiability and extensibility of the model bad. 3) Though we have to make distinction between contexts and values shown by sensors, as sensors are the means of capture contexts, we sometimes confound contexts with sensors and make tightly coupled model. In this paper we propose an aspect-oriented context modeling technique for embedded systems, in which we show a strategy to model embedded systems that have context-dependent nature. The aspect-oriented context model is developed in the early stage of development and can be used as a reference model for software development in the following stages.

1. Introduction

With the development of information technology, embedded systems become more and more large and complicated. They have to control variety of hardware, they are required to show higher performance and reliability, they are connected to network to communicate with other equipments and information systems, and so forth.

These tendencies change the development style of embedded systems. Compared with the development style of business applications, that of embedded systems used to be ”implementation centric”. However, as embedded systems become larger and more complicated, we have to put much emphasis on early stages of development, and to that end, modeling becomes more and more important for the development of embedded systems.

One of the characteristics of embedded systems is context dependency. Here, context means any environment in which the system is operated. For example, it reacts to the context change (e.g. if it becomes dark, then turn on the light) and its behavior is constrained by the context (e.g. even if heating button is pressed, it does not start heating if the temperature is too high). These context-dependent systems have three important conceptual parts. One part consists of sensors that show some values that reflect phenomena in the external worlds, one part consists of logics to decide contexts based on these sensor values, and the other part consists of internal processings that are triggered or constrained by these determined contexts. For example, in vehicle, sensors for a hand-brake and a shift-gear detect their position, then the system determines whether the vehicle completely stops or not, and internal processing abort services that interfere driving.

As these three parts are essential for context dependent systems, requirements for such systems are defined in terms of these three parts, and the software architecture of the system reflects them. Furthermore, as each part relates to different properties of the system, each part requires different kinds of check and verification. Therefore it is desirable to develop model in the early stage that capture these three parts, and utilize the model as a reference model for architecture design, checking/verification, and so on. One of the problems in modeling such context dependent systems is that it is not straightforward to localize these three parts, as we have to handle various kinds of contexts such as logical contexts and physical contexts, and they have complicated relationships.

In this paper, we propose an aspect-oriented context modeling for embedded systems that have these context-dependent nature. In the technique we show a strategy to model embedded systems utilizing aspects. In our technique, we introduce a new modeling element, ”inter aspects relation”, that explicitly define the relationship among as-

pects.

2. Context Modeling

In this paper, we call any environment in which a system is operated as **context**. There are various kinds of contexts, and they may change time to time, and place to place. For example, considering the systems for automobile, the followings are some examples of possible contexts:

- the temperature, humidity, brightness around the vehicle.
- the position, velocity, acceleration of the vehicle.
- the engine status, door position, gear position.
- the intensity of radio field of cell phone and wireless LAN, the number of satellites they can capture for GPS measurement.
- the services available from outside information systems (for example, each gas station, parking lot, or toll booth may provides different services).

Systems cannot recognize contexts directly, and they utilize **sensors** as the means to capture these contexts, and judge and conjecture the current contexts. We may use multiple sensors to capture a context, same sensors may be used to capture different contexts and there may be multiple means to capture a certain context. Therefore, it is important to distinguish between contexts from values shown by sensors.

A **context-dependent system** is a system that has internal processing that depends on its contexts. The followings are the typical characteristics of the internal processing of a context-dependent system:

- **reaction:** the system reacts to their external events and/or context changes, i.e. contexts trigger the internal processing. e.g. if the temperature arises, then the system starts cooling.
- **constraint:** the system's behavior is constrained by the contexts, i.e. the system determines the type of the behavior, considering the current context. e.g. even if the operator sends cooling command, the system does not start cooling, if the temperature is too low.

In analyzing and designing context-dependent systems, we have to model contexts and relationships among contexts and internal processing. We call these modeling activities as **context modeling**.

3. Example: Onboard Software Management System

In this section, we introduce Onboard Software Management System (OSMS for short), and examine the problems in context modeling.

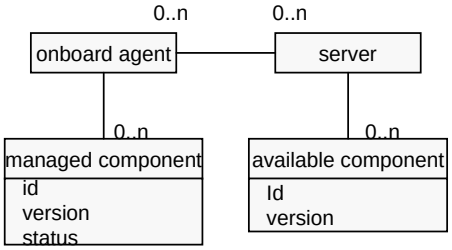
3.1. Basic Functionality

OSMS is the system to manage software components that are installed on the vehicle onboard system. Recently the size of software for onboard system becomes larger, it is required to manage these software components like those on personal computers and workstations. Namely, it becomes necessary to update and replace the software components when a problem arises and/or newer versions are released.

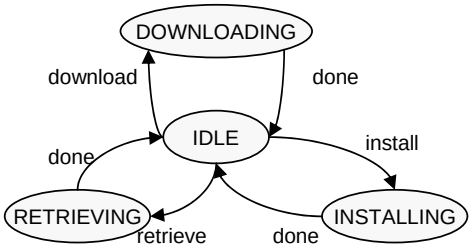
In this paper, we examine the OSMS that has the following functionalities.

- OSMS can communicate with central servers by cell phone. It can also communicate with local servers that are placed at local points such as gas stations and toll booths by wireless LAN. Even if OSMS communicates with the central server, when it comes into the area where wireless LAN is available, the system can switch the communication media.
- In the central servers and local servers, there are software components for downloading. Each server may have a different set of software components, as it is difficult to deliver the same version of software to all the servers simultaneously.
- OSMS can get the list of available software components from servers, via cell phone or wireless LAN. Based on the operation of the driver or passengers, OSMS can download the software components, and installs them on the vehicle onboard system.

Figure 1 shows the basic model of OSMS. Class "server" resides on the server side, and stores and manages the available software components. Class "available component" is the information about software component that is placed on the server. Class "onboard agent" resides on the vehicle side, and manages downloading and installing of the software components. Class "managed component" is the information about the available software component for vehicle sides, and its status indicates that it is downloaded or not, and it is installed or not. The "onboard agent" downloads the information about the available components (retrieve), downloads the components (download) and installs the downloaded components (install).



(a) class diagram



(b) state diagram for onboard agent

Figure 1: Basic Model of OSMS

3.2. Context Dependency

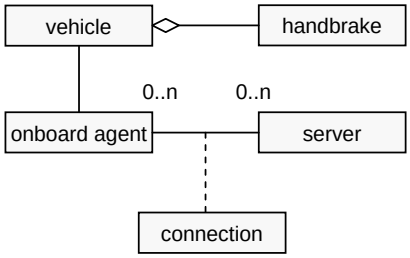
OSMS has the context-dependent features. The followings are examples of the context dependencies:

- Even if driver/passengers specify install command, ”onboard agent” does not start installing if the vehicle is not in the safe condition (typically vehicle is running). It is because installing new software components may causes dangerous situation during the driving.
- If the vehicle enter the unsafe condition (typically vehicle start running) during installation, ”onboard agent” aborts installation.
- If the communication path between ”onboard agent” and ”server” is disconnected during downloading, ”onboard agent” preserves the status, and resumes downloading if the connection is established again. As connected server may be changed (because vehicles are moving), ”onboard agent” has to check whether or not the same component is available from the new server.

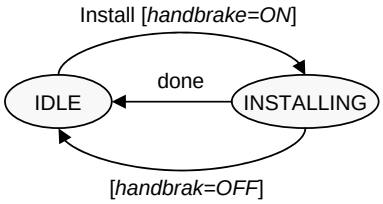
As described above, OSMS has to consider their contexts related to vehicle status (running or not), communication status (connected or disconnected) and servers status (current set of downloadable components).

3.3. Problem

Figure 2 shows an example of OSMS model (part) considering the context dependency. Here, in order to model contexts, class ”vehicle”, ”handbrake” and ”connection” are



(a) class diagram



(b) State diagram for onboard agent (installing part)

Figure 2: OSMS Model considering Context

introduced, and ”onboard agent” detects vehicle status by means of ”handbrake”.

Based on this model, we examine problems of context modeling.

various kinds of contexts

OSMS depends on various kinds of contexts: Server status is a logical context in which we are interested in available components provided by the connected server. On the other hand, communication status is a physical context that relates to connection/disconnection and bandwidth of the communication path.

Though these contexts are captured from different points of view, it is not straightforward to model them independently, because they relate each other. For example, the server status model includes logical connections to servers, and communication status model includes physical connections to servers. These two kinds of connections are different aspect of vehicle-center relationship, i.e. they are not exactly the same but they have relationship. It is common for context dependent systems to handle multiple contexts, and these contexts share the same objects, or they relate to different aspects of the same objects, and this makes the context modeling difficult.

strong coupling with context

As shown in Figure 2 (b), internal model directly depends on contexts. If we change the referred contexts, that causes direct effects on internal model. In the current version of the model, the state diagram refers the status of ”handbrake” in the guard condition. If we try to refer other

conditions (such as vehicle velocity and the type of operator), we have to change the model.

As stated in the previous item, a context dependent system handles different contexts that relate each other, we can easily introduce complicated relationship among contexts and internal processing. For example, internal processing may detect disconnection with the server in terms of server context (logical disconnection) or communication context (physical disconnection), but partial order of these two disconnections can change depend on how to model two contexts and relationship among them.

confound sensors with context

In Figure 2, we do not distinguish between contexts and sensors, and the state diagram of ”onboard agent” directly depends on sensors (”handbrake”). It is not good for the modifiability and extensibility of the model, because there are multiple ways to capture the same context (such as handbrake, brake, gear position, and velocity), and we may change the means, even if we are to capture the same context.

4. Aspect Oriented Context Modeling

In this section, we examine the application of aspect-oriented technology to context modeling.

4.1. Basic idea

Aspect-oriented technology is a technology, in which we use aspect as a new mechanism for modularity, that corresponds to a certain concern [1]. In this paper, we utilize the aspect as a mechanism to modularize models; Entire model consists of one or more aspects, and each aspect includes a class diagram and state diagrams relate to the concern.

The basic idea of applying aspects to context modeling is as follows:

- As we have to handle various kinds of contexts, we utilize aspects to model each context. Each aspect is defined from a certain point of view appropriate for each context. As a same modeling target (such as vehicle) may appear in multiple contexts, it is suitable use aspect rather than ordinary UML package.
- As mentioned in the previous section, it is not desirable to make models for ”internal processing”, models for ”context” and models for ”sensor” tightly coupled. We also utilize aspects to disjoint these three categories of model, and introduce three ”stereotypes” for aspect (<<process aspect>>, <<context aspect>> and <<sensor aspect>>) to explicitly express these three categories of model.

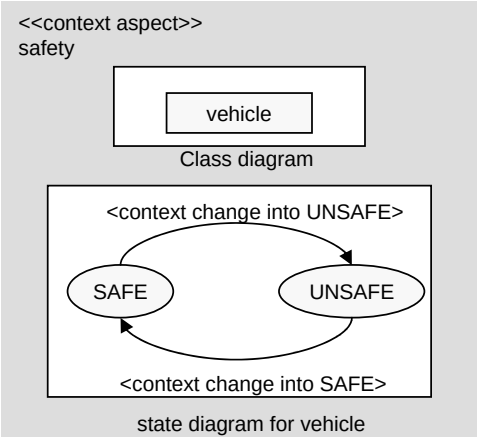


Figure 3: Context Aspect (safety)

- In aspect-oriented programming, mechanisms that relate different aspects are pre-defined as the language constructs. However, in aspect-oriented modeling, there are not common and widely accepted mechanisms yet. Therefore, we explicitly model the relationships among aspects using modeling element ”inter aspects relation”.

4.2. Application of Aspect-Oriented Technology

We will explain how we apply aspect-oriented technology to context modeling, using OSMS example.

various kinds of contexts

We define each context as an aspect, in which we view the system from a specific point of view. These aspects have stereotype <<context aspect>>.

- Vehicle status is about the safety of installation, and modeled in ”safety aspect”.
- Server status is about the logical connection among ”onboard agent” and ”server”, and modeled in ”server aspect”.
- Communication status (connection and bandwidth) is about the physical status of communication, and modeled in ”communication aspect”.

Figure 3 shows a context aspect (”safety” aspect). This figure denotes that ”safety” aspect of stereo type <<context aspect>> includes a class diagram (that defines class ”vehicle”), and a state diagram for ”vehicle” class.

strong coupling with context

In order to avoid strong coupling among contexts and internal processing, we define them as independent aspects. Aspects that represent context have stereotype <<context

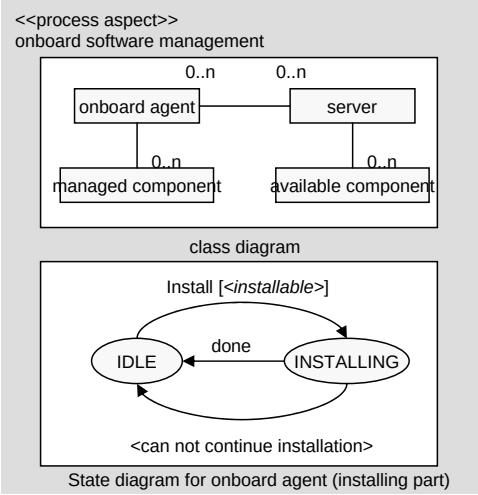


Figure 4: Process Aspect (onboard software management)

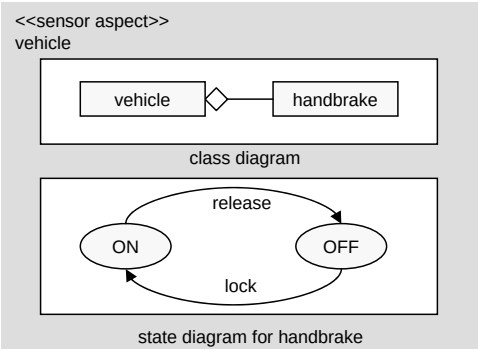


Figure 5: Sensor Aspect (vehicle)

aspect>>, and aspects that represent internal processing have stereotype <<process aspect>>. If we directly refer contexts in process aspect, internal processing and context become strongly coupled. Therefore, in process aspect, we use abstract context to avoid the direct coupling. In OSMS example, we use abstract names ”installable” and ”can not continue installation” to avoid direct coupling. These abstract names will be related to other aspects using inter aspects relation explained shortly.

Figure 4 shows a process aspect (”onboard software management” aspect).

confound sensors with context

In order to distinguish contexts from sensors, we also define them as independent aspects. Aspects that represent sensors have stereotype <<sensor aspect>>.

Figure 5 shows a sensor aspect (”vehicle” aspect). Note that ”vehicle” is appeared in ”safety” aspect and ”vehicle” aspect, but it is obvious that these two classes are modeled from different point of view.

4.3. Inter Aspects Relations

In order to model entire system, we have to relate independently defined aspects[10]. In aspect-oriented programming, we define each aspect using language constructs such as ”advice”, and the weaver merges aspects into a program.

In modeling field, on the other hand, there are no common and widely accepted mechanisms yet. Therefore, in this paper, we examine what kinds of relationships among aspects are required in context modeling, and introduce a modeling element ”inter aspects relation” to explicitly define relationships among aspects.

Based on OSMS example, we observe that relationships listed in Table 1 are necessary for context modeling.

Table 1: Inter Aspects Relation appeared in Context Modeling

response	Context changes in ”context aspect” trigger the activation of processes in ”process aspect”. E.g. if it becomes ”SAFE” in ”safety aspect”, then abort ”INSTALLING”.
condition	Processes in ”process aspect” refer contexts defined in ”context aspect”, and determine the behavior based on those. E.g. even if ”Install” event comes in ”onboard software management” aspect, it does not go into ”INSTALLING” if it is ”UNSAFE” in ”safety” aspect.
abstraction	Context aspect determines contexts by referring the contexts in other ”context aspects” and status in ”sensor aspects”. E.g. ”safety” aspects determines ”SAFE” and ”UNSAFE”, based on status in ”handbrake” aspect.
consistency	Keep consistencies among aspects. E.g. if physical connection in ”communication aspect” becomes disconnected, then delete the logical association between ”onboard agent” and ”server”.

In our context modeling, we introduce inter aspects relations to express these relationships. An **inter aspects relation** is a modeling element by which we define the dependency among modeling elements in aspects. Each inter aspects relation imports modeling elements defined in two or more aspects, and relates these elements using relationship shown in Table 2.

Figure 6 is an example of inter aspects relation, in which relations between ”onboard software management” aspect and ”safety” aspect are defined. Inter aspects relation is described as rectangle, and ”imports” the data elements from two or more aspects. (In the figure, we abbreviate class dia-

Table 2: Basic Relationship used in Inter Aspects Relation

refer	A model element in one aspect is defined in terms of other model elements in other aspects. E.g. a guard condition is defined in terms of attributes and states defined in other aspects.
trigger	A behavior in one aspect triggers the behaviors in other aspects. E.g. update of attribute value, invocation of method and firing of transition in one aspect trigger those in other aspects.

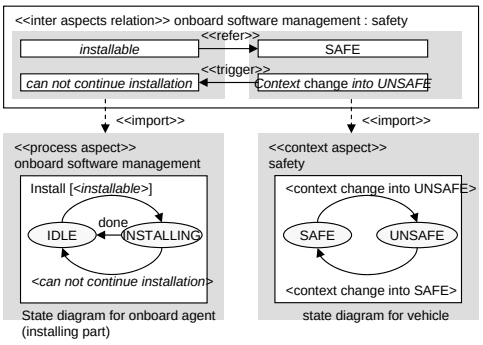


Figure 6: Inter Aspects Relation (response, condition)

gram in ”onboard software management” aspect and ”safety aspect”). In the rectangle, there defined dependencies (refer, trigger) among data elements imported from these aspects. Here, ”installable” in ”onboard software management” aspect is defined in terms of conditions defined in ”safety aspect (”SAFE” or not), and the transition from UNSAFE to SAFE in ”safety” aspect triggers the transition from INSTALLING to IDLE in ”onboard software management” aspect.

Figure 7 shows another example in which relations between ”safety” aspect and ”vehicle” aspect are defined. The states of ”vehicle” (SAFE, UNSAFE) are defined in terms of the status of ”handbrake” (ON, OFF).

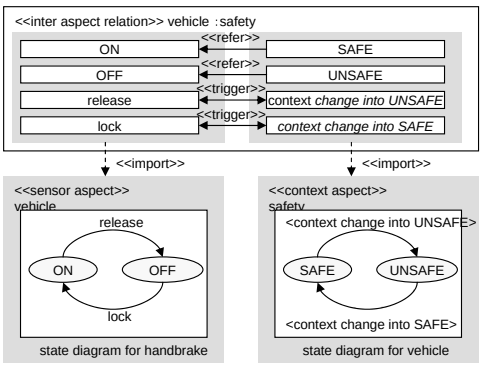


Figure 7: Inter Aspects Relation (abstraction)

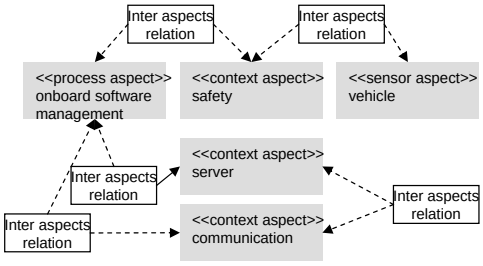


Figure 8: Overall Relations among Aspects for OSMS

The transitions in ”vehicle” aspect and transitions in ”safety” aspect triggers each other: As ”vehicle” aspect is concrete side, and ”safety” aspect is abstract side, the transition is firstly about to fire in ”vehicle” aspect. Before actually fire transition, it triggers the transition in ”safety” aspect, and if it actually fires, then it triggers back the transition in ”vehicle” aspect. This kind of interactions is necessary to keep the consistency between concrete side and abstract side, because in ”safety” aspect, there may be some guard conditions are given, and triggered transition may not fire. Figure 8 shows the overall structure of OSMS example.

Note that, the inter aspects relation is introduced in order to explicitly define relationships among aspects in order to overview the entire structure, as dependency design (how to reduce the strong coupling among models for internal processing, models for context and models for sensor) is one of the most important issues in context modeling. Therefore, concrete mechanisms to realize ”refer” and ”trigger” relations are not our main focus.

5. Discussion

In this section, we discuss a few technical issues.

5.1. Context Modeling

We examine advantages and disadvantages of our aspect-oriented context modeling.

- As we have to handle variety of contexts, it is good to model them from different points of view. On the other hand, if we develop aspects without careful consideration, the entire model becomes just a gathering of small pieces of aspect, and inter aspects relations become complicated. It is important to work out a strategy for deciding aspects and developing entire structure of the model.
- As each aspect basically does not depend on other aspects, we can avoid strong coupling among internal processing, contexts and sensors. This is a good characteristic, because in developing context dependent systems, we need many iterations before fixing

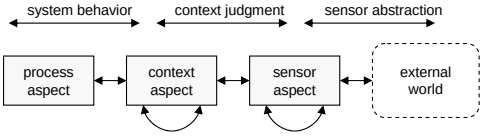


Figure 9: Reference Model for Checking Validity

what contexts should be captured and what sensors should be used to determine contexts.

For example, consider the situation in which we want to refine SAFE condition to be determined not only by handbrake but also by gear position. In our model, we just add ”gear” class to ”vehicle” aspect, and modify ”refer” relation for SAFE condition (Figure 7). We do not need to change the ”onboard software management” aspect.

- As pointed out previously, relationships among contexts and sensors are not one to one. Consider the situation in which the same sensor is used to capture different contexts, it is difficult to model sensor from one point of view. For example, if we use vehicle velocity to determine safety, it may be abstracted as safe speed and unsafe speed. However, if we use vehicle velocity to determine the time to distance, such abstraction does not work. If we use aspects, we can model them straightforwardly, i.e. define two aspects, one for safety, the other for time calculation.

5.2. Checking the Validity

It is important to check the validity of developed model. In order to effectively check the model, it is indispensable to have the good strategy of checking. There are some techniques to check the validity, such as review, test, simulation and model checking technologies[5], and we have to apply right technologies to right place. In context dependent systems, each conceptual part has different sets of properties to be checked, and we can use our context model as a reference model for checking the validity.

Figure 9 shows a reference model for checking the validity based on our context model. This reference model shows that there are three categories for checking the validity in context-dependent system. The followings are the rough sketch of the checking strategy, based on this reference model.

- **sensor abstraction**

  In the context model, ”sensor aspect” and relationship (via inter aspects relation) among them relate to abstraction of the external world.

  Here, we have to decide how to abstract sensor value (e.g. a sensor may be modeled to have two values ON and OFF) and relationship among them (e.g. sensorA and sensorB do not have value ON at the same time).

  To check if the abstraction is adequate or not is difficult, because it is a matter of recognition of the real world. For example, a sensor may be modeled to have three values ON, OFF and ERROR, instead of ON and OFF. To determine which abstraction is adequate must be checked by test (i.e. actually operate the system in the real world), or by expert’s review.

  On the other hand, it is relatively easier to check whether or not the model correctly reflects the determined abstraction. For example, using model checker, we can check important property such as ”sensorA never becomes ON whenever sensorB is ON, vice versa”.

- **context judgment**

  ”Context aspects”, relationship among them and relationship among ”context aspects” and ”sensor aspect” relate to judgment of context. Here we decide contexts to be captured, and determine these contexts by means of other contexts and sensors.

  It is difficult to check if contexts are correctly identified. For example, even though we identify the ”SAFE” and ”UNSAFE” situations, we cannot determine the adequateness of this abstraction, because it is again a matter of recognition of the real world. We use review or test to check the adequateness of them.

  On the other hand, it is relatively easier to check if the model correctly reflects the definition of each context. For example, using model checker, we can check important property such as ”whenever handbrake is released, it never judged as SAFE”.

- **system behavior**

  ”Process aspects” and relationship among ”process aspects” and ”context aspects” relate to the context dependencies of the main functionalities.

  We can check if the interaction among processes and contexts (i.e. reaction to the contexts and constraint by the contexts) are defined correctly or not by review, test, simulation and model checker. As ”process aspects” do not directly refer the ”sensor aspect”, but refer ”context aspects”, it is expected that we can reduce the number of states to be checked.

5.3. Related Work

We compare our work with other works.

- Aspect-oriented technologies are studied not only for programming phase[3, 7], but also wide range of software development. There pointed out that we have to

handle variety of concerns[12] in software development. There are variety of proposal to apply aspect-oriented technology to upper phases of software development such as architectural design[6], design considering non-functional requirements[2, 9], and analysis of requirements on product-lines[8]. We focus on context modeling issues, and utilize aspect-orientation to model context-dependent systems in the early stage.

- In regard to weaving at modeling level[4], there is no common and widely accepted approach yet. In this paper, we do not argue "model weaver", but we introduce inter aspects relation to explicitly define relations among aspects. Our intention is to support context modeling, in which we want to reduce strong coupling among sub models. Therefore, these models are just for human understanding, and no concrete mechanism is given.
- In our approach, we define three types of aspects, process, context and sensor, each of them corresponds to important conceptual part of context dependent system. This technique has the similarity with domain bridge technique[11], but has the following differences:
  - In stead of ordinary package or module, we utilize aspects by which we can handle different abstraction of the same design target.
  - We introduce inter aspects relation, and in that we capsulate dependencies among aspects. This dependency design is different from ordinary bridge and adopter.
- As embedded systems have interactions with real world, it is indispensable to have many iterations before fixing the model. Therefore, it is important to make the model loosely coupled so as to we can easily modify some part of the model without causing modification to other parts.

  On the other hand, embedded systems have to be operated under the limited resources, such as CPU performance and memory size. Therefore they prefer non-redundant implementation. Our modeling approach is good for modifiability and extensibility, but it will introduce redundant code, if we straightforwardly reflect the model structure into implementation structure. We need to examine the implementation technique for our model to avoid the efficiency problem.

6. Conclusion

In this paper, we picked up the context dependency, that is one of the important characteristics of embedded systems, and examine problems in context modeling. Based on that, we have proposed an aspect-oriented context modeling technique to develop context model that has appropriate structure. We will refine the technique based on case studies, not only considering modifiability and extensibility, but also considering other quality attributes such as reliability.

References

[1] http://aosd.net

[2] Georg, G., Ray, I. and France, R. : Using Aspects to Design a Secure System, Proceedings of the Eighth IEEE international Conference on Engineering of Complex Computer Systems (ICECCS '02), 2002.

[3] Harrison, W. and Ossher, H.: Subject-Oriented Programming - A Critique of Pure Objects, Proceedings of OOPSLA'93, 1993.

[4] Ho, W., Jezequel, J., Pennaneac'h, F. and Plouzeau, N.: A Toolkit for Weaving Aspect Oriented UML Designs, Aspect Oriented Software Development (AOSD) 2002, 2002.

[5] G.J., Holzmann, :The model checker SPIN, "IEEE Trans. on Software Engineering, Vol. 23, No. 5, 1997, pp. 279-295.

[6] Katara, M. and Katz, S.: Architectural Views of Aspects, Aspect Oriented Software Development (AOSD) 2003, 2003.

[7] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, M., Lopes,C.V., Loingtier, J. M. and Irwin, J.: Aspect-Oriented Programming, Proceedings of ECOOP '97., 1997.

[8] Kishi, T. and Noda, N.: Aspect-Oriented Analysis for Product Line Architecture, 1st Software Product Line Conference (SPLC1), 2000.

[9] Noda, N. and Kishi, T.: On Aspect-Oriented Design - An Approach to Designing Quality Attributes-, Proceedings of 6th Asia-Pacific Software Engineering Conference, p230-237, 1999.

[10] Noda, N. and Kishi, T.:On Aspect-Oriented Design Model, Foundation of Software Engineering, 2003 (in Japanese).

[11] Shlaer, S. and Mellor, S.: Object Lifecyles: Modeling the World in States, Prentice Hall, 1992.

[12] Sutton, Jr., S.M. and Rouvellou, I.: Concerns in the Design of a Software Cache. Workshop on Advanced Separation of Concerns in Object-Oriented Systems, Nov. 2000.