Mapping Concern Space to Software Architecture: A Connector-Based Approach

Jing (Janet) Liu
Dept. of Computer Science,
Iowa State University
226 Atanasoff Hall, Ames, IA 50011
+1 (515) 294-2735
janetlj@cs.iastate.edu

Robyn R. Lutz
Dept. of Computer Science,
Iowa State University
226 Atanasoff Hall, Ames, IA 50011
and Jet Propulsion Laboratory/Caltech
+1 (515) 294-3654
rlutz@cs.iastate.edu

Jeffrey M. Thompson
Research & Development
Guidant Corporation
4100 Hamline Ave North,
St. Paul, MN 55112
+1 (651) 582-5739
Jeffrey.Thompson@guidant.com

ABSTRACT
Concern modeling plays an important role in software design, implementation and maintenance. Hyperspace has provided a strong conceptual framework to separate concerns in multi-dimensional levels. The contribution of this work is to create an architectural element, called a concern connector, to support the implementation of hyperspace in the architectural design phase. The paper makes three basic claims for this idea. First, using concern connectors allows the scope of each hyperslice in a certain concern dimension to be defined and stored. Second, the concern interactions within each hypermodule can be specified in the concern connectors. Third, the association of concern modeling with this distinctive architectural element improves the flexibility of concern maintenance and evolution during the development process. To test these claims the paper investigates the use of concern connectors in a real-world architectural model. The results show how concern connectors implement concern modeling in the architectural design.

Categories and Subject Descriptors
D.2.2 [**Design Tools and Techniques**]: Modules and interfaces, Object-oriented design methods; D.2.11 [**Software Architecture**]: Languages, Patterns

General Terms
Design, Languages.

Keywords
Hyperspace, Software Connector, Aspect, Feature Interaction.

1. INTRODUCTION
Concern spaces provide guidance to software engineers in handling system modularization in both design and implementation phases; they also give a good reference for maintenance and evolution [16]. Concerns are defined to be "those interests which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders" [5]. An example of a concern is data integrity. A concern space describes the idea of separating and segregating the set of currently considered concerns over the set of units that constitute the software in the domain of discourse [16].

The work described here builds on Hyperspace [11], a general conceptual framework for multi-dimensional separation of concerns. Hyperspace is a concern space with a well-defined multi-dimensional structure, as described in Section 3.

Our research is motivated by the need to separate and model concerns in large software systems, especially in safety critical product lines. For safety-critical systems, maintenance and verification of cross-cutting concerns are essential. Product lines must also support the addition of new features. For example, a product line consisting of several different pacemakers and defibrillators might add a new feature to let doctors view parameters remotely.

To address these issues, this paper introduces a specialized connector, called a concern connector, to define the scope of separated concern pieces in the architectural design. A concern connector is an architectural element that specifies the scope of concerns in the architectural design phase. We show how the Hyperspace approach can be mapped into the architectural design, specifically, how hyperslices [11] can be modeled and presented using the concern connectors. We also show how to integrate a set of hyperslices to form a hypermodule [11] using concern connectors. The integration is not simply a hierarchy of concern connectors but rather a new concern connector with the scope and the interactions (the correspondence relationships among hyperslices) explicitly specified. The paper also discusses how concern connectors support adding new features to a product line, performing on-going safety analysis, and using aspect oriented programming.

The contribution of our work is two-fold. First, concern connectors extend the modeling ability of the architecture model from traditional architectural concerns to multiple concerns with a focus on safety. Second, concern connectors help bridge the gap

between concern modeling and aspect oriented programming (AOP) in large systems so that the work of concern separation and identification can be largely preserved in the architecture design. Consequently concern connectors help guide subsequent implementation and maintenance.

The rest of the paper is organized as follows. Section 2 addresses related work and introduces an industrial application. Section 3 gives the definition and overview of concern connector. Section 4 describes the connector-based approach, including the hyperslice and hypermodule modeling in the architecture. Section 5 discusses the implications of concern connectors for concern maintenance and evolution. Section 6 provides a brief conclusion.

2. RELATED WORK & APPLICATION

The work described here is a natural extension of the architectural connector notion supporting the realization of Hyperspace [11] in the architectural domain. Similar work includes Hyper/J [13], which implements the Hyperspace notion in the code level in Java; ConcernBASE [3], which extends UML to realize viewpoint language to describe concern space, extending that to SADL(the Structural Architectural Description Language); Hyper/UML [14], which maps Hyperspace into different UML elements to be used in the feature modeling in the model based development; Theme/UML[2], which captures separated concerns through UML design model encapsulation and composition; Dynamic Hyperspaces [1], which uses connectors for hyperslice composition to provide a dynamic view of Hyperspace; the Hyperspace generalization using meta-models [9], which generalizes the Hyperspace notion to support artifact language in UML; and modeling crosscutting concerns using software connectors [7], which provides UML support for a connector to model component interactions, as well as architectural concerns that crosscut the boundaries of components.

Our approach differs from these studies in that we use the specialized architectural element, concern connector, to define and maintain the scope of a hyperslice (piece of concern), and to support the integration of hyperslices. The connector specification is general enough to be extended to any specific architectural description language. Another difference is that the concern connector provides a systematic and scalable way to allow the AOP referencing of any part in a large system.

The running example in the paper is the software architecture of a defibrillator product line. A defibrillator is an embeddable medical device designed to monitor and regulate the beating of the heart when the heart is not pacing at a normal rate. Its major functions include detecting abnormal cardiac rhythms (including tachycardia and bradycardia, which are fast and slow abnormal heart beats, respectively), and applying therapies (e.g., stimulating the heart with an electrical pulse or shock). The therapies are applied to two chambers of the heart: ventricle and atrium [4].

Within this domain, concerns typically capture a wide variety of features and required properties of interest. Here we use two safety-related concerns to motivate and explain our method:

(1) Tachycardia therapy (for a fast heartbeat) in the ventricle (called ventricular tachycardia therapy) should always have priority over tachycardia therapy in the atrium (called atrial tachycardia therapy)

The rationale behind this safety concern is that, because the ventricles supply approximately 80% of the circulatory capacity, ventricular tachycardia is more life-threatening than atrial tachycardia. Thus, ventricular tachycardia must always have priority.

(2) In bradycardia therapy (for a slow heartbeat), the defibrillator should always give a pulse to the heart when no heartbeat is detected during a certain time interval.

The rationale behind this safety concern is that when the heart has bradycardia symptoms, the lack of heartbeat for a certain period is life threatening and thus should be treated with an electrical pulse.

The two concerns are so important that they need to be captured and maintained through different development phases. They are also both cross-cutting in nature in that they can affect several blocks in the architecture, as described below.

3. CONCERN CONNECTOR

In order to capture and maintain the concerns in a software architecture, we define the notion of concern connector. The template in Table 1 details the information that should be provided for each concern connector. Section 4 describes an example.

The architectural part of this approach is based on the following essential features of a software architecture description language (ADL): components (including the interfaces), connectors and configurations [15]. An architectural configuration is a connected graph of components and connectors describing architectural organization. We denote each component, lower-level sub-component, or higher level subsystem in such an ADL as an architectural block.

Table 1. The general concern connector representation.

Concern description	The specific concern it is capturing
Implementation Set	The set of architectural block(s) whose interactions are relevant to the current concern and their sub-blocks that directly receive the inputs or produce the outputs involved in this interaction.
Interface Set	The set of architectural block(s) that provide inputs to the block(s) in the implementation set. These architectural blocks are specified as concern_connector_name.block_name if they refer to another concern connector.
Rules	i) The interactions of the blocks and sub-blocks in the Implementation Set if the interaction is under the current concern ii) The constraints on the blocks that are connected by multiple concern connectors
Rules Enforcement	The places to enforce the rules. When not implemented, it serves to record the scope related to certain concern(s); once the rules are implemented, the units affected by the current concern register themselves here for possible future maintenance updates.

We next give a brief overview of the Hyperspace framework. Hyperspace's approach [12] divides concerns into a set of disjoint groups, each of which is called a *dimension of concern*. The modeling pieces in a Hyperspace are hyperslices and hypermodules. A hyperslice collects all the units that address the same concern. Hyperslice are declarative complete in that they "declare everything to which they refer" [11]. A hypermodule provides the context for a set of interacting hyperslices to be integrated by means of a set of composition rules (that specify how they correspond to each other) [11]. Thus, a hypermodule addresses multiple concerns and can form a natural building block for the software system. (Note that in this paper we do not consider concerns that are hard to measure via architectural blocks, such as "time to market".)

4. MAPPING CONCERN SPACE TO SOFTWARE ARCHITECTURE

The units in the Hyperspace definition are essentially the architectural blocks in the architectural structure. Our focus is on mapping the existing concern space into the architectural model.

4.1 Mapping Hyperslices

We first show that the union set of the *Implementation set* and the *Interface set* in a concern connector can capture all the units related to that concern and is *declarative complete*, thus forming a hyperslice addressing that concern.

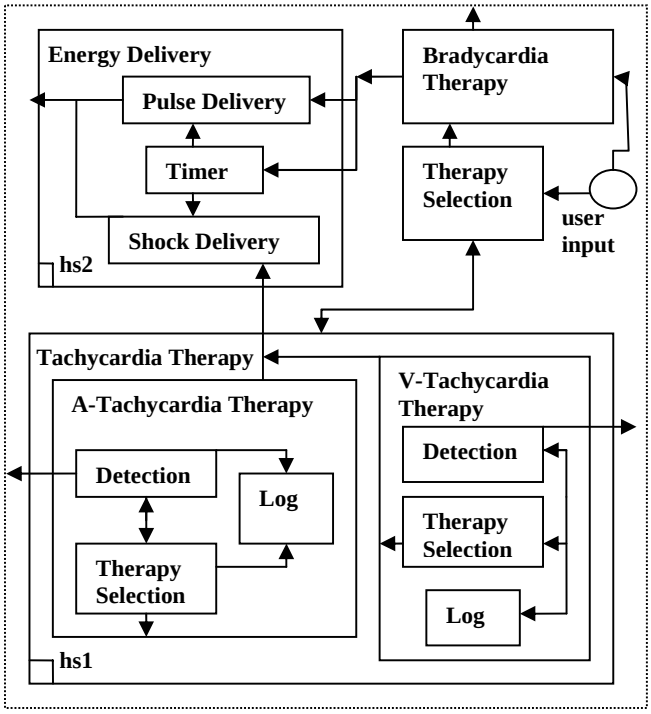

Figure 1. The architectural configuration of the Defibrillator core.

To demonstrate this claim, we first look at what needs to be in a hyperslice for a concern, namely the units that encapsulate that concern. Thus, those blocks whose interactions are contained in the current concern, as well as the sub-blocks of those blocks that either receive the inputs or produce the outputs involved in this interaction, should be included, since they are all part of the implementation of the current concern. This set is defined in the Implementation Set in the concern connector.

Table 2. The concern connector defining hyperslice (1).

Concern Description	Tachycardia therapy in the ventricle should always have priority over tachycardia therapy in the atrium
Implement-ation Set	Tachycardia Therapy(A-Tachycardia Therapy(Detection, Therapy Selection), V-Tachycardia Therapy(Detection, Therapy Selection))
Interface Set	Therapy Selection
Rules	When the Therapy Selection block selects Tachycardia Therapy, the Tachycardia Therapy block uses V-Tachycardia Therapy sub-block's output unless it has no output, in which case it uses A-Tachycardia Therapy sub-block's output
Rules Enforcement	Should be implemented as a new connector to arbitrate between A-Tachycardia Therapy and V-Tachycardia Therapy

According to the definition of declarative completeness, hyperslices **"**must declare everything to which they refer**"** [11]. In an architectural configuration, these blocks are the ones that invoke the blocks in the Implementation Set. In other words, since their output provide input to the blocks in the Implementation Set, they form the Interface Set in the concern connector. Since we are not concerned at this point with their internal structures, they are referenced as concern_connector_name.block_name (if they are included in the Implementation Set of another concern connector), or simply as block_name. In the latter case these units should update to the form of concern_connector_name.block_name once they are included by another concern connector.

Note that the blocks that are influenced by the outputs of the blocks in the current Implementation Set are not included, as they do not contribute to the current concern. Similarly, those blocks that provide input to the blocks in the Interface Set are not explicitly included, as they are implicitly included when we refer to the concern connector in which they are defined in their Interface Set.

Table 3. The concern connector representing hyperslice (2).

Concern Description	In Bradycardia Therapy, the defibrillator should always give a pulse to the heart when there is no heartbeat detected for a certain interval of time.
Implement-ation Set	Energy Delivery(Pulse Delivery, Timer), Bradycardia Therapy
Interface Set	user input, Therapy Selection
Rules	When in Bradycardia Therapy, if Timer timeouts while Bradycardia Therapy block has not detected any heartbeat, Pulse Delivery should give pulse
Rules Enforcement	Should be implemented by a controller component in Energy Delivery

Furthermore, a concern connector provides the rules that the units must satisfy when they merge under the same concern. In this way the concern connector captures the information needed to define a hyperslice. Note that the rules here are specified informally. However, in practice, formal logics or organization-specific languages are used.

To better describe the workings of a concern connector, we use a simplified architectural configuration describing the defibrillator core (see Fig. 1). (The "user input" is referencing some other part of the system not shown in this graph, such as the GUI.) Table 2 shows the concern connector for the first hyperslice of concern labeled as (1) in Section 2. (Note that "Log" is not included because it does not directly receive inputs in this interaction.)

The architectural representation of this concern connector is shown as a square labeled "hs1" in Tachycardia Detection at the bottom left of Fig. 1. It will become a real architectural component or connector once implemented.

Table 4. A general Hypermodule modeling scheme.

Concern Description	Subset of the concerns modeled in the sub-slice concern connectors
Implement-ation Set	The union of the *Implementation Sets* of the concern connectors of the hyperslices
Interface Set	The union of the *Interface Sets* of the concern connectors of the hyperslices excluding the *Implementation Set* of this hypermodule
Rules	1) When the *Implementation Set* of any of the hyperslices overlap: i) if the concerns are mutually exclusive, specify when to shift from one to the other ii) if there are contentions among concerns, specify the priority among them; iii) specify any dependencies among concerns; else, apply "merge" relations to simply integrate those concerns on the units; 2) If the *Implementation set* of some hyperslices overlap with the *Interface set* of other sub-slices, specify the "binding" relation between them: the units in the *Implementation set* should fulfill the requirements for the same units in the *Interface set*; no contention nor mutual exclusion should ever occur.
Rules Enforcement	Should be realized in the overlapping blocks specified above.

Table 3 shows the concern connector for the second hyperslice of concern labeled as (2) in Section 2. ("Shock Delivery" is not included because it does not directly receive inputs in their interaction.) The architectural representation of the concern connector is again a square, here labeled "hs2" in the Energy Delivery.

4.2 Mapping Hypermodules

The way to map a hypermodule into the architecture is very similar to the way in which a hyperslice is mapped into the architecture. Essentially, a hypermodule is a higher-level hyperslice that addresses part or all of the concerns from the hyperslices constituting it [13]. We illustrate the general form of this representation in Table 4.

When modeling hypermodules, the concern connector captures the concern interactions within each hypermodule by systematically integrating the connectors representing the hyperslices constituting it. The rules specified in the table are based on some of the correspondence relationships in [11]. Other rules to explore for hyperslice integrations include those in [16].

5. MAINTAINING CONCERN INFORMATION AS THE SYSTEM EVOLVES

We here briefly sketch two benefits of concern connectors beyond the design phase: to map to aspects and to support product-line evolution.

One application of concern maintenance is mapping concern connectors to aspects in aspect oriented programming (AOP). The work in [6] has already described the mapping of general connectors to *pointcuts* and *advice* in AspectJ [8]. Table 5 gives a simple example of how a concern connector can be mapped to elements of AspectJ. (The example is shown in an abstract manner).

The location of the concern connector suggests when an aspect should be introduced. The concern connector also allows an easier and more natural derivation of aspects. Each concern connector for hyperslices can map to a single aspect because each addresses one concern. The concern connector for hypermodules may need to be divided into multiple aspects, each of which addresses one or more rules within it.

Concern connectors preserve concern information not only for AOP or the programming phase, but also for other system tasks such as static testing or safety analysis. By maintaining the scope in each connector we can more easily select an appropriate and consistent scope for a model checker to verify. Since each connector maps a hyperslice which is declarative complete, the derived model will also be declarative complete. In addition, the rules specified in each connector can be mapped to formal properties.

Since the concern connector serves as a fixed architectural element providing information about where and how to enforce the rules regarding one concern (hyperslice) or a group of concerns (hypermodule), it is helpful in concern maintenance and evolution, especially in product lines.

The advantage of concern connectors for product lines is that product-line features (e.g., optional variations) can be treated as concerns. Known feature interactions may also be resolvable by the rules specified in the concern connectors. For example, to find the interaction of a new concern with an existing concern we build the hypermodule of the two, as shown in Table 4. Comparison of their Implementation Sets and Interface Sets helps find interactions of the new concern with existing concerns. Similarly, if a concern (or feature) is deleted, reference to the

Implementation Set of its concern connector and to the blocks shown in the Rules Enforcement helps identify the affected units. We anticipate that concern connectors will also assist us in our safety analysis of evolving product lines. First, connectors maintain a clearly defined scope for the safety concerns. Second, the "Rule Enforcement" specification for concerns captures design safeguards that need to be maintained even as the systems change.

The recommendation in [11] that "when new units are added, they must be added in hyperslices", here translates into a recommendation that when new architecture blocks are added, they should be connected to some concern connector.

Table 5. Mapping the concern connector of concern (1) into an aspect.

Concern Connector	AOP
Implementation Set: Tachycardia Therapy(A-Tachycardia Therapy(Detection, Therapy Selection), V-Tachycardia Therapy(Detection, Therapy Selection))	To capture all the message passing in the Implementation Set, we use *cflow()* pointcut[8] to capture a chain of messages (one invokes the other), or use *call()* pointcut to capture a single message, e..g., cflow(call(V_Tachycardia_Detection.output)) denotes all messages involved in the Tachycardia_Detection block's output
Interface Set: Therapy Selection	Since Therapy Selection is the output of interest, we model it in an *args()* pointcut, e.g., args(Therapy_Selection)
Rules: When Therapy Selection block selects Tachycardia Therapy, the Tachycardia Therapy block uses the output from V-Tachycardia Therapy's sub-block's; in the case that it has no output, the block uses the output from A-Tachycardia Therapy's sub-block	Use around advice [8]; try to proceed with V-Tachycardia Detection's output; if there is a null exception, proceed with A-Tachycardia Detection's output instead
Rules Enforcement: should be implemented as a new connector to arbitrate between A-Tachycardia Therapy and V-Tachycardia Therapy	The scope of influence of this aspect should be the classes in the package of A-Tachycardia Therapy and the package of V-Tachycardia Therapy

6. CONCLUSION

This paper defined a concern connector to assist in mapping concern spaces into an architectural design. The paper showed how concern connectors modeled both the scope of hyperslices and the concern interactions within hypermodules. An application to a real-world embeddable medical device illustrated the use of concern connectors. Finally, the paper discussed the advantages of concern connectors for maintaining concern information as systems evolve. This is especially important in safety-critical product lines such as the application described here.

7. ACKNOWLEDGEMENT

This research was supported by the National Science Foundation under grants 0204139 and 0205588.

8. REFERENCES

[1] Chitchyan, R., and Sommerville, I. Composing Dynamic Hyperslices. *Workshop on Correctness of Model-based Software Composition (ECOOP 2003).*

[2] Clarke, S. and Walker, R. J. Towards a Standard Design Language for AOSD. *Proc. 1st Int'l Conf. Aspect-Oriented Soft. Dev.*. Enschede, The Netherlands, 2002, 113-119.

[3] Crettaz, V., Kandé, M. M., Sendall, S., and Strohmeier, A. Integrating the ConcernBASE Approach with SADL. *UML 2001*. 166-181.

[4] Ellenbogen, K.A. and Wood M.A. *Cardiac Pacing and ICDs*: 3rd Edition. Blackwell Science, 2002, 415-424.

[5] IEEE Architecture Working Group. *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*. IEEE Std 1471-2000, IEEE, 2000.

[6] Kandé, M. M., Kienzle, J., and Strohmeier, A. *From AOP to UML: Towards an Aspect-Oriented Architectural Modeling Approach.* TR 200258. Swiss Fed. Ins. Tech., 2002.

[7] Kandé, M. M., and Strohmeier, A. Modeling Crosscutting Concerns using Software Connectors. *ASoC3.* Tampa Bay, Florida, 2001.

[8] Laddad, R. *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning publications, 2004.

[9] Lohmann, D., and Ebert, J., A Generalization of Hyperspace Approach Using Meta-Models. *The 2003 AOSD Early Aspects Workshop (AOSD-EAWS'03).* Boston, MA.

[10] Mehta, N. R., Medvidovic, N., and Phadke, S., Towards a Taxonomy of Software Connectors. *Proc. ICSE 2000.* Limerick, Ireland, 2000, 178-187.

[11] Ossher, H., Tarr, P. *Research Report: Multi-Dimensional Separation of Concerns in Hyperspace.* IBM Research Report 21452(96717), 1999.

[12] Ossher H., Tarr, P. Multi-Dimensional Separation of Concerns and the Hyperspace Approach*. Proc. Symp. Sw. Arch. & Component Technology*, 2000.

[13] Pekilis, B. R. *Multi-Dimensional Separation of Concerns and IBM Hyper/J.* TR., Bell Canada Software Reliability Laboratory, U. of Waterloo, 2002.

[14] Philippow, I., Riebisch, M., and Boellert, K. The Hyper/UML Approach for Feature Based Software Design. *The 4th AOSD Modeling with UML Workshop.* SF, CA, 2003.

[15] Shaw, M. and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline.* Prentice-Hall, 1996.

[16] Sutton, S.M., and Rouvellou, I. Modeling of Software Concerns in Cosmos. *Proc. 1st Int'l Conf. Aspect-Oriented Soft. Dev.*. Enschede, The Netherlands, 2002, 127-133.