Pattern Language for the Modeling Practices of MATLAB and Simulink

HIROKI ITOH, Mitsubishi Electric Corporation
YASUO HOSOTANI, Mitsubishi Electric Corporation

Model-based development (MBD) is a powerful method for designing control algorithms. MATLAB and Simulink are adopted as a standard tool for MBD. However, it could be an obstacle especially for beginners to handle the tools, which could make it difficult for them to easily achieve their purpose. Therefore, collecting and sharing technical information for Simulink modeling is a critical issue for enhancing algorithm development. This paper introduces the art of MATLAB and Simulink modeling and provides insights using a pattern language. This knowledge is derived from practical experience. The pattern language consists of 10 patterns that focus on the basic practices and manners of modeling. They will help developers to conduct effective and efficient MBD with MATLAB and Simulink.

Categories and Subject Descriptors: D.2.11 **[Software Engineering]**: Software Architectures—Patterns

General Terms: Pattern Language, Model Base Development, MATLAB, Simulink

Additional Key Words and Phrases: Modeling, Simulation

ACM Reference Format:

Itoh, H. and Hosotani, Y. 2023. Pattern Language for the Modeling Practices of MATLAB and Simulink. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 30 (October 2023), 11 pages.

1. INTRODUCTION

Model-based development is a technique that can improve the effectiveness of developing new products. It has been adopted especially in the automotive industry and space industry and has the advantage of simulating a new control algorithm. Simulink, working on MATLAB, is a worldwide tool for realizing model-based development. It can be used to design an algorithm by adding blocks and connecting them intuitively using a graphical user interface and to visually confirm the result with a graph.

To make use of the advantages of model-based development, it is necessary to investigate the overview of Simulink and acquire the operation to create a model. Therefore, exploiting the advantages of Simulink will be a long way for developers with less experience. Further, disseminating the modeling experience to other engineers will be a challenge.

There are some guidelines and research that illustrate the patterns for designing models with Simulink. The MathWorks Advisory Board (MAB) provides guidelines for the modeling control algorithm [MAB 2020]. It explains the styles of implementing Simulink and Stateflow models and describes the model architecture: six layers that constitute a hierarchy. MathWorks also published a whitepaper on the best practices for application to specific standards, namely, ISO 26262 [Moore and Lee 2019] and AUTOSAR [Jaffry and Keener 2020]. Besides MathWorks, Whalen et al. proposed a model structure for efficient verification by classifying models into four groups and changing combinations according to a verification case [Whalen 2014]. In addition, Jaskolka et al. provided four guidelines to improve the modularity of a Simulink model [Jaskolka 2020]. The design policies for constructing models have been discussed among the developers' community, though there is a lack of a report on patterns for acquiring valid attitudes and operations of modeling.

Herein, we introduce the knowledge of modeling with MATLAB and Simulink as a pattern language, which transmits the practical expertise of utilizing Simulink effectively. The knowledge will enable developers to eliminate the burden of acquiring the approach to work effectively using the tool, which normally requires a certain amount of experience. This paper provides an overview of our pattern language, the list and structure of patterns, and detailed descriptions of each pattern.

Author's address: H. Itoh, 8-1-1, Tsukaguchi-hommachi, Amagasaki, Hyogo, Japan, email:Ito.Hiroki@dr.MitsubishiElectric.co.jp; Y. Hosotani, 8-1-1, Tsukaguchi-hommachi, Amagasaki, Hyogo, Japan, email: Hosotani.Yasuo@eb.MitsubishiElectric.co.jp

The rest of this paper is structured as follows. Section 2 describes the landscape of our pattern language and a list of patterns. We discuss the characteristics of the patterns related to model-based development in Section 3. Section 4 provides an example of adaptation with a story. Section 5 details each pattern. Finally, Section 6 concludes the paper.

2. OVERVIEW OF PATTERN LANGUAGE

The pattern language consists of 10 patterns for utilizing MATLAB and Simulink more effectively. All the patterns are listed in Table 1.

We categorize our patterns into two groups—one group deals with patterns in creating models, and the other, in conducting model-based development.

Table 1 The List of the Patterns

No.	Name	Abstract
Group 1: Create a Model		
1	Simple First	Create a simple model in the first step and sophisticate it later.
2	No Dialog	Modify properties of blocks by not using dialogs, but using panes.
3	Easy Experiment	Make a trial with a small experimental model when the usage of blocks is obscure.
4	Runnable Sample	Refer to samples in order to understand how to embed unfamiliar blocks.
5	Expert's Advice	Do not hesitate to call for help to resolve the problem faced.
6	Imitation of Shape	Observe models others create and imitate the usage of blocks and the implementation of logic.
Group 2: Conduct model-based development		
7	Minimum Setting	Configure the minimum definitions of model interfaces for coupling models smoothly.
8	Restored Result	Restore the result of each simulation to confirm that there are any differences between before and after modification.
9	Visualized Message	To convey the design intention, represent points of modeling visually.
10	History of Errors	Store information on errors and resolutions to avoid another time-consuming debug.

In Fig. 1, we describe the structure of the pattern language. The dotted lines indicate the relationship between patterns.

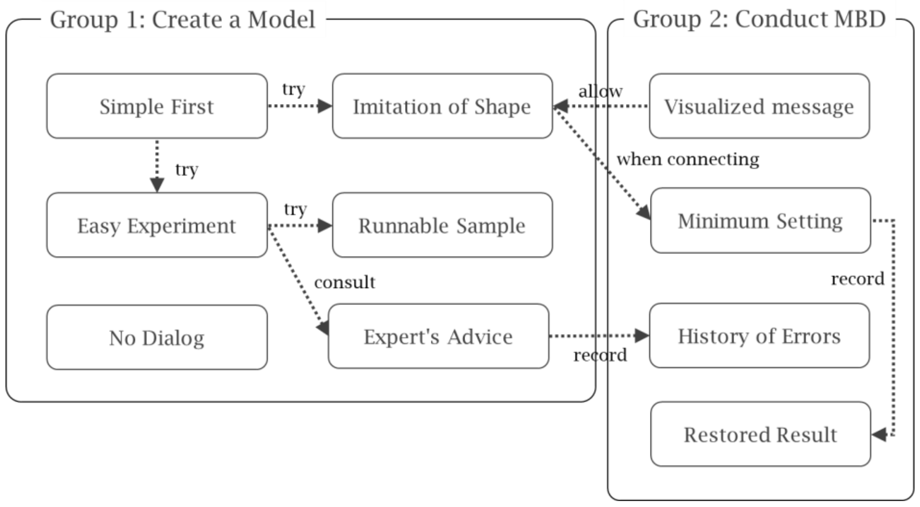

Fig 1. Structure of the Patterns

3. SCENARIO

In this section, we describe an example of applying the pattern language to development through a scenario.

SCENARIO: Alice's Modeling

Alice has joined a model-based development project. However, Alice has no experience with it and is using Simulink for the first time. She is extremely anxious.

It is determined that she works with Bob—her senior colleague. First, Alice endeavors to create a model in a **Simple First** way. She sets aside the definition of an interface's data type according to **the Minimum Setting** lesson.

Alice vaguely recognizes the use of basic blocks. However, there is a block that she has not encountered before. It is tough for her to imagine its usage even after consulting a detailed reference. Therefore, she simulates **Runnable Sample**s provided by MathWorks, confirms the behavior and creates a small model for a trial as an **Easy Experiment**. Thus, Alice becomes used to various types of blocks.

Alice faces a problem related to the usage of one block during modeling. She inquires Bob about it. However, he has little experience in working with that block. Therefore, she requests an **Expert's Advice** to reveal the usage by sending a message with the result of **Easy Experiment**s. This benefits her in terms of eliminating a trial-and-error period. She constructs the whole of her model based on **Imitation of Shape** from the model that was earlier created by Bob. His model is sophisticated to adapt to the model architecture properly and represents the intention of design by a **Visualized Message**.

Alice's first model is implemented to some measure. She configured the settings of blocks using the Model Data Editor with the consciousness of **No Dialog**.

On completing the model, Alice faces an error, which she finds difficult to comprehend; however, she could remove it by consulting the **History of Errors** accumulated by her team. Finally, testing shows that her model satisfies the requested specifications. She exports the simulation logs and saves them as the **Restored Result** in future regression tests.

Thus, Alice successfully completes her first modeling. Bob is satisfied with her good performance.

4. PATTERNS

In this section, we describe the patterns listed in Table 1. The details are as below.

Group 1: Create a Model

The patterns in Group 1 are the basic ones used in creating a model. It has seven patterns, and their details are given below.

Simple First

Abstract

Make a simple model first and avoid pursuing the detailed setting of blocks and the exact definitions of parameters from the beginning.

Context

We are going to start modeling. The control model often has many parameters to adjust its algorithm. We should set the model parameters and find proper values adapted for the given situation.

Problem

Parameters with a detailed setting can be time intensive. The Simulink Parameter, which is a concept for parameterization, has many settings, such as data type and auto code generation.

Force

Simulink has a concept that treats parameters effectively as a Simulink Parameter and a data dictionary.

Parameters should be defined for future developments. We consider introducing these concepts and elaborating parameters in models. However, an early data definition may incur additional costs because it is necessary to check the detailed configuration for data type and code generation.

Solution

Keep in mind to make a simple model at first. The details of a model should be set aside until the later stage of modeling.
For example, set parameters as a MATLAB Variable when we implement the function from scratch. After determining the model's logic, replace these parameters as a Simulink Parameter and define them in suitable data dictionaries. In addition, do not insert the logic for a program module in considering an algorithm such as the measure against division by zero.

Result

It will prevent us from spending time because of unnecessary settings on a first logical verification. The detailed configuration should be considered at an appropriate time: Parameterization and definition in Data Store, for instance.

No Dialog

Abstract

It may be effective to edit the properties of models without the dialog appearing by a double-click.

Context

We are creating a model and changing the parameters of many blocks.

Problem

We change the parameter of a block by the dialog, such as a block parameter dialog. The operations can be tedious, involving the repetition of display and closing of the dialog if many blocks should be configured.

Force

It is intuitive to change model parameters by a dialog displayed by a double-click. Replacing the operation in another way may cause a feeling of strangeness. However, in the case of many blocks, displaying the dialog for each block could be tedious and time-consuming.

Solution

Utilize the Model Data Editor positively when we edit the parameters of the blocks. It enables us to change the parameters of multiple models in a single pane. In addition, the properties of a block may be changed without displaying the Block Property dialog. The property can be configured by the Property Inspector pane, which is embedded with a model window. Inserting a block could also be realized without a dialog named the Library Browser. Use the quick insert menu by double-clicking on a model canvas to add a block instead of the dialog.

Result

The pattern can compress the time of modeling by eliminating the dialog operations. It can help us to reduce the stress in modeling by ensuring smooth editing.

Easy Experiment

Abstract

Conduct an easy experiment if the behavior of an adopted block is unknown.

Context

We create new logic in **Simple First** way and attempt to add a block that has been rarely used or has never been used.

Problem

To realize a required behavior by referring to documents on the Internet will not be effective if we do not know the blocks. Or, the algorithm will be implemented in another way if we do not use the said block, so that the readability of the model may decrease as a result of using substitute blocks.

Force

There may be resistance to building a model for testing behaviors since model creation is needed. Developers experienced in general programming languages, such as C or Java, tend to hesitate to implement a trial model; preparing the environment and logic is necessary for implementing such a model.

Solution

Try to simulate the block and consult the result if we do not recognize the behavior of a block. Simulink can confirm the behavior of the block easily by merely pressing the simulation button. Check the configuration of the block and consult documents about block specification if the result is different from the estimated one. It can be an effective process to cycle editing settings and check behavior. **Simple First** is preferred to recognize the function of the block.

Result

Taking advantage of this pattern will require less time to examine the block usage. The model created by an experiment would be useful in asking for **Expert's Advice** because it could illustrate the troubled situation as a concrete example.

Runnable Sample

Abstract

Run a sample model if the usage of a block or a subsystem is uncertain.

Context

We are deploying new logic in a model. It seems that one block can satisfy desired logic. However, its usage is not clear.

Problem

Searching documents to learn the usage will consume much time. In addition, unexpected behavior may arise if we adopt a block despite the ambiguity of its usage.

Force

It is difficult to guess the operation of the block based on only some text. Although the information is available on the MathWorks website, the installation of a block will be challenging without a precedent.

Solution

First, check the reference on the website. If a sample is available, try to simulate it. This will be helpful to understand the way of using and configuring the block as we can confirm its behavior via simulation.
In another case, we sometimes instruct team members about how to create a model in a certain way. In this situation, we create a sample model by ourselves. This sample will be helpful to convey information regarding the design.
The model assembled in an **Easy Experiment** might be eligible as a sample model. That is an **Easy Experiment** is personally conducted to check the behavior of an unknown block by trial and error and would be a **Runnable Sample** for other members after achieving that confirmation. If there is a significant point about a model structure, it is effective to build the sample model using a **Visualized Message**.

Result

A **Runnable Sample** enables us to recognize the behavior of a block and assemble the logic rapidly. In addition, it helps improve the modeling quality by sharing the art of the design with other team members.

Expert's Advice

Abstract

Ask the expert to provide advice if there is a question in creating a model.

Context

We are deploying new logic in a model. However, we do not know how to implement it or there are errors in the simulation during an **Easy Experiment**.

Problem

It could be expensive to find the solution for the problem which we have never faced and we may be forced to give up implementation of the logic in the worst case.

Force

We might try to resolve the problem by ourselves because of the worry that asking questions could reveal their lack of skill. There may be no advisors around them in the first place. Developers with experience in another programming language may not discard the bias that the answer could be found by a web search relatively easily. Actually, not much information about Simulink is available because the user community is smaller compared to other programming languages.

Solution

Ask questions to the experts by utilizing MATLAB Answer, the open questions and answer site, or MathWorks Software Maintenance Service. In this pattern, an expert is not a senior modeler in your team but a specialist in the Simulink communities around the world. If the query is a frequent one, similar answers can be found in MATLAB Answer. Asking MathWorks directly could be available as long as the maintenance service is valid.
In requesting, it is necessary to clarify and organize the question to convey the problem accurately. An **Easy Experiment** can be efficient for this purpose. A small sample model arranges the expectations and the result of finding a solution to the problem via a simulation.

Result

The pattern can reduce the time required for searching the manner of modeling to achieve the resolution. Furthermore, it can provide relief during modeling, which is often solitary work, as reliable partners are available when a problem arises. Once the inquiry leads to a resolution, team members will be motivated to approach experts and eliminate the waste of time for investigation. The acquired knowledge is shared as one of **History of Errors** after achieving the goal. There may be associates who face the same problem at a later stage in modeling. However, it is important to be patient when seeking expert advice, as responses may not be immediate due to the experts' time constraints.

Imitation of Shape

Abstract

Imitate the layout of another model when designing a new one.

Context

We create new logic in **Simple First** way. The new logic is not simple, and it seems difficult to implement it without sufficient consideration of the whole design.

Problem

We might create a poor-quality model by conducting ad hoc implementation and because of the inability to devise a proper constitution. In addition, the model may also be inconsistent with the others created by the other team members.

Force

A model is readable for people since it is represented graphically. It is easier to judge whether a model resembles what is to be made.

Solution

Imitate the appearance of a model if it realizes a similar function to what you want to create. The meaning of "imitate" is to adopt how to locate blocks and connect signals if they seem to match the new one, not just to copy and paste a logic. The Simulink model is represented graphically, and it is easier than the models of general programming languages.
When preparing for imitation, consideration of model structure in advance will be effective if several models with similar characters are planned to be created. The model quality can be improved if a suitable structure is defined and shared with team members before implementation.

Result

The pattern can eliminate confusion in structuring the model and accelerating modeling. In the review process, reviewers can focus on the logic of a model because its structure is familiar to them. Keep a model simple with **Minimum Setting** to combine models smoothly.

Group 2: Conduct model-based development

The patterns in Group 2 are used for conducting model-based development. Group 2 consists of four patterns, which are described below.

Minimum Setting

Abstract

Coordinate the minimum interface definitions to couple models smoothly.

Context

We created several models in **Simple First** way and try to combine them to realize a large function.

Problem

If there is no adjustment of the interface definition among models, errors will occur because of the dimension or data type mismatch. Hence, the dimension and data type should be configured. However, it is burdensome to deal with all the input/output signals. When one definition has to be changed, we need to consider reviewing the other ones.

Force

Defining a data type and dimension is tedious work. This task tends to be put off because it is not the implementation of the logic itself. Simulink provides an option named "Inherit: auto" that resolves the data type automatically. This option may be useful if the scale of a function is comparatively small. Nevertheless, some definitions for the ports should be specified, especially for code generation.

Solution

Specify the data type and dimensions of a model as needed. These definitions for the input and output ports should be set minimally to bind models. A policy should be formulated to determine how strictly these definitions should be set within a team.

Result

The pattern enables us to couple models successfully because their interfaces are matched. Furthermore, the time cost of the task is eliminated by adopting the minimum interface definitions. However, if overly strict rules are established regarding **Minimum Setting**s, it can become burdensome for all team members.

Restored Result

Abstract

Restore the result of a simulation, including input data, to confirm the model's behavior before modification for regression testing.

Context

We are modifying some models. It is necessary to check if the models can provide the intended result and if there is no unpredicted impact.

Problem

Understanding the behavior of the entire task is a complex task. Especially, the level of difficulty increases if the architecture of models is large or complicated.

Force

We are concerned that modification may cause the failure of the total simulation. The frequent failure of attempts despite sensitivities represents this difficulty. It is complicated to investigate behavioral changes by chasing signals. In general, it needs some resources to store data for regression testing. However, MATLAB and Simulink could collect simulation logs and access stored data easily.

Solution

Store the input data and the results of the simulation at the end of the behavioral confirmation and compare the simulation result with the stored one after modification using the same inputs. We can test whether the behavior after modification is as expected. It is relatively simple to accumulate data with Simulink than with general programming languages.

Result

Restored results can prevent unexpected behaviors accompanying model updates. The pattern can be utilized in many situations, which may involve the impacts of modification.

Visualized Message

Abstract

Express the message visually to indicate the object of modeling design or settings.

Context

We are creating a model. Some information should be shared with team members; however, the information will be complicated if it is expressed only using text.

Problem

It may be necessary to draft a long comment to describe the object of design. However, if the comment is added to the model, it may get buried in many blocks of the model because generally, the color used for text and blocks is black.

Force

Writing is a common mode of sharing information such as the object of design and note of implementation. Therefore, we may provide all information as text unless we are aware that it should not be so.

Solution

Consider the color and location of blocks when we need to convey the model consideration.
One example of conveying information effectively is to indicate a time-series logic flow visually; that is, the earlier processes are on the left side of a model and the later ones are on the right. Another example is to arrange blocks in a vertical line to indicate that they are performed in parallel.
It might be a good idea to change the background color of a model to gray if its logic and structure should not be modified unintentionally. The change in background color draws our attention. Similarly, it is also effective to indicate tentative implementation by setting the color of the block to red.

Result

The **visualized message** conveys the intention of the design and model logic without using a long text. However, even if we build models with attention to their appearance, other members may not realize implicit messages. To prevent this from happening, formulate a team's rule about the meaning of each color. Once all team members are aware of the rule, we can adopt **Imitation of shape**, which will reinforce the power of patterns.

History of Errors

Abstract

Record the information about errors faced during modeling to share the resolution with team members.

Context

We created a model and are trying to compile it. Unfortunately, the model generated an error during compilation. We checked the error message shown in the dialog; however, we have never seen that error and are unaware of its resolution.

Problem

Resolving the error and realizing a successful simulation may take a long time. Sometimes, Simulink outputs an error message that is difficult to comprehend.
We can inquire with the team members about the error. The situation may be improved if we remember the solution. However, we may have forgotten the experience and resolution.

Force

We may hesitate to ask other team members for a solution because we feel that the team member looks busy at work or may doubt the skill if we ask for a solution to an easy problem.
We tend to be satisfied when the error is eliminated and may simply switch to the next work, however, when the team members encounter the same error, they would struggle to resolve it.,

Solution

Once the error is resolved, record the details of the error and its resolution for future reference. For example, when an inconsistency error of data type happens, it may be useful to set the configuration of data type to "Inherit: Inherit via back propagation".
If we can access old records, they can solve the same problem when facing the same error again. The same error can often be repeated in the team because the team members create models with a similar context.
In the beginning, recording is conducted by setting a team rule. When we call for the **Expert's Advice**, put together the information, including the identity of the expert who answered it.

Result

The pattern enables us to return to modeling by removing an error smoothly. On the other hand, we need to set aside time to record and organize information of errors for a contribution to our team.

5. DISCUSSION

This section discusses the characteristic points of our patterns and their relationships with other previous pattern languages. We have organized them in Table 2. Here, we describe the features of each pattern in the model-based development domain. In addition, the table lists the characteristic points, with the two segments that they are related to, namely, "Problem and Solution" and "Force".

The first segment is "Problem and Solution". A problem or solution itself could be unique for model-based development. For example, the **No Dialog** pattern treats the burden of setting parameters and comments as a problem in modeling and provides a solution that can be defined by the operation panes of a modeling window. Its problems and solution are specific to model-based development topics.

The other segment is "Force". In this case, the problem and solution of a pattern are not necessarily characteristic of model-based development. However, the external factor around the context of a pattern reinforces the significance of the problem and the effectiveness of its solution. For example, the **Easy Experiment** pattern describes a relatively general problem and solution, which suggests creating a simple test module and checking its behavior against an unclear specification. In contrast, there is a unique force that encourages the adoption of the solution because the model can be easily simulated in Simulink. This is the factor that makes the pattern specific to the model-based development domain.

Next, we explain the relationships between our patterns and the ones proposed in previous works. To compare them, we would like to introduce two pattern languages. One is "Fearless Change" [Manns and Rising 2004], which describes ways of tackling by applying a new idea or technology to organizations. The other is "Learning Patterns" [Iba 2009], which introduces patterns to acquire new knowledge autonomically. We selected these pattern languages because they mention effective ways to absorb a new concept, which conforms with ours.

Table 2 lists several patterns that have a relationship with other pattern languages. These patterns commonly have characteristic points related to model-based development in the Force description. We suppose this is because their problems can be adapted to various situations as they are aimed at solving generic problems. The **Restored Result** pattern has no relationship with these pattern languages, however, it would enhance the regression testing practice in software development.

Table 2 Characteristic Points to Model-based Development and Related Patterns in Other Pattern Languages

No.	Name	Characteristic Points to Model-based Development			Related Patterns	
			Problem Solution	Force	Fearless Change	Learning Patterns
1	Simple First	Simulink has many settings, and the compilation overhead tends to be greater than in general programming languages.		✓	Step by Step	Start small, Let it grow
2	No Dialog	The settings of the blocks are achieved by dialogs intuitively in Simulink. But it is not necessarily efficient.	✓			
3	Easy Experiment	Simulink can easily simulate models and give the results to be consulted.		✓	Just do it	First Steep
4	Runnable Sample	There is not much information about simulation on the Internet compared with general programming languages.		✓		Prototyping
5	Expert's Advice	There is not much information about simulation on the Internet compared with general programming languages.		✓	Guru on Your Side	
6	Minimum Setting	There are many items for configuring models. On the other hand, Simulink allows auto-resolution for settings.	✓			
7	Imitation of Shape	Simulink could describe the logic of a model visually.		✓		Mimic Learning
8	Restored Result	It is easy to not only store the simulation result but compare each result in the Simulink.		✓		

9	Visualized Message	Simulink could represent information with a diagram as well as a word.	✓			
10	History of Errors	There is not much information about errors on the Internet compared with general programming languages.		✓	Hometown Story	

6. CONCLUSION

This paper describes the pattern language to make MATLAB and Simulink modeling more efficient. We hope that the pattern language can be utilized for beginners to gain knowledge of modeling with the tools. We can continue to obtain feedback by applying the patterns to an actual development and discussing its validity with developers to obtain new insights.

REFERENCES

David Jaffry and Holly Keener. 2020. 10 Best Practices for Deploying AUTOSAR Using Simulink.
Jason Moore and John Lee. 2019. 11 Best Practices for Developing ISO 26262 Applications with Simulink.
Mary L. Manns and Linda Rising. 2004. Fearless Change: Patterns for Introducing New Ideas.
MathWorks Advisory Board (MAB). 2020. Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow, Version 5.0.
Michael W. Whalen, Anitha Murugesan, Sanjai Rayadurgam and Mats P. E. Heimdahl. 2014. Structuring Simulink Models for Verification and Reuse.
Monika Jaskolka, Vera Pantelic, Alan Wassyng and Mark Lawford. 2020. Supporting Modularity in Simulink Models.
Takashi Iba, Toko Miyake, Miyuko Naruse and Natsumi Yotsumoto. 2009. Learning patterns: A Pattern Language for Active Learners.