Software Design Specification

for

Lightweight Educational Assistant (LEA) the Lego Robot

**Version 1.0 approved**

**Prepared by FLAIR**

**Stacey Joseph**

**Lauren Massey**

**Jessica Updegrove**

**Embry-Riddle Aeronautical University - SE555**

**February 16, 2015**

**Table of Contents**

[Introduction (Stacey)](#h.cep4prs9p39j)

[Document Outline](#h.bbii8gguuoza)

[Document Description](#h.o8ybwpngqm3m)

[Introduction](#h.2mg06nusy2ip)

[System Overview](#h.fjmzguw17hbt)

[Design Considerations (Stacey)](#h.rrh9cd65d6cr)

[Assumptions and Dependencies](#h.hjq7xh8yx7ht)

[General Constraints](#h.9pt60xbb3w7d)

[Goals and Guidelines](#h.ihilv16eq7pd)

[Development Methods](#h.6ozqjryqzbdb)

[Architectural Strategies (Lauren)](#h.2nz389mobpet)

[System Architecture (Jess)](#h.bs9p99pahy1w)

[Subsystem Architecture](#h.d5mtocw7mtcc)

[Policies and Tactics (Lauren)](#h.o3b4akha5z0)

[Detailed System Design (Jess)](#h.9vljq96y6koy)

[Glossary](#h.lkapnqg7r1l7)

[Bibliography](#h.1j6iwd5jj14)

# **Introduction (Stacey)**

LEA (Lightweight Educational Assistant) is a Lego Mindstorms robot that will realize a behavioral design pattern called “strategy.” The interactions performed by LEA are organized by the tiers in which the request for information begins. LegoRobot is the class that implements the Lejos API. The main program will utilize the speak, hear and move functionality for the RobotInteract. Within, the Hear function are word sets that are anticipated inputs from the voice recognition subsystem. The established input will be confirmed by the system interaction with the user.

## **Document Outline**

Here is the outline of the proposed template for software design specifications. Please note that many parts of the document may be extracted automatically from other sources and/or may be contained in other, smaller documents. What follows is just one suggested outline format to use when attempting to present the architecture and design of the entire system as one single document. This by no means implies that it literally is a single document (that would not be my personal preference):

* Introduction
* System Overview
* Design Considerations
  + Assumptions and Dependencies
  + General Constraints
  + Goals and Guidelines
  + Development Methods
* Architectural Strategies
  + Strategy design pattern
* System Architecture
  + component-1 name or description
  + component-2 name or description
* Policies and Tactics
  + policy/tactic-1 name or description
  + policy/tactic-2 name or description
* Detailed System Design
  + module-1 name or description
  + module-2 name or description
* Glossary
* Bibliography

The above outline is by no means exclusive. A particular numbering scheme is not necessarily required and you are more than welcome to add your own sections or subsections where you feel they are appropriate. In particular you may wish to move the bibliography and glossary to the beginning of the document instead of placing them at the end.

The same template is intended to be used for both high-level design and low-level design. The design document used for high-level design is a "living document" in that it gradually evolves to include low-level design details (although perhaps the "Detailed Design" section may not yet be appropriate at the high-level design phase).

The ordering of the sections in this document attempts to correspond to the order in which issues are addressed and in which decisions are made during the actual design process. Of course it is understood that software designs frequently (and often fortunately) don't always proceed in this order (or in any linear, or even predictable order). However, it is useful for the purpose of comprehending the design of the system to present them as if they did. Frequently, one of the best ways to document a project's design is to keep a detailed project journal, log, or diary of issues as they are mulled over and bandied about and to record the decisions made (and the reasons why) in the journal. Unfortunately, the journal format is not usually organized the way most people would like it for a formal review. In such cases, for the purpose of review, the journal can be condensed and/or portions of it extracted and reorganized according to this outline. However, if this is done then you need to choose whether to update and maintain the design document in the journal format, or the formal review format. It is not advisable to try and maintain the design document in both formats. (If you have an automated method of converting the journal into a formal document, then this problem is solved.)

## **Document Description**

Here is the description of the contents (by section and subsection) of the proposed template for software design specifications:

### **Introduction**

Provide an overview of the entire document:

* Describe the purpose of this document
* Describe the scope of this document
* Describe this document's intended audience
* Identify the system/product using any applicable names and/or version numbers.
* Provide references for any other pertinent documents such as:
  + Related and/or companion documents
  + Prerequisite documents
  + Documents which provide background and/or context for this document
  + Documents that result from this document (e.g. a test plan or a development plan)
* Define any important terms, acronyms, or abbreviations
* Summarize (or give an abstract for) the contents of this document.

Note:

For the remaining sections of this document, it is conceivable (and perhaps even desirable) that one or more of the section topics are discussed in a separate design document within the project. For each section where such a document exists, a reference to the appropriate design document is all that is necessary. All such external (or fragmented) design documents should probably be provided with this document at any design reviews.

### **System Overview**

Provide a general description of the software system including its functionality and matters related to the overall system and its design (perhaps including a discussion of the basic design approach or organization). Feel free to split this discussion up into subsections (and subsubsections, etc ...).

# **Design Considerations (Stacey)**

This section describes many of the issues which need to be addressed or resolved before attempting to devise a complete design solution.

## **Assumptions and Dependencies**

Describe any assumptions or dependencies regarding the software and its use. These may concern such issues as:

* Related software or hardware
* Operating systems
* End-user characteristics
* Possible and/or probable changes in functionality

## **General Constraints**

Describe any global limitations or constraints that have a significant impact on the design of the system's software (and describe the associated impact). Such constraints may be imposed by any of the following (the list is not exhaustive):

* Hardware or software environment
* End-user environment
* Availability or volatility of resources
* Standards compliance
* Interoperability requirements
* Interface/protocol requirements
* Data repository and distribution requirements
* Security requirements (or other such regulations)
* Memory and other capacity limitations
* Performance requirements
* Network communications
* Verification and validation requirements (testing)
* Other means of addressing quality goals
* Other requirements described in the requirements specification

## **Goals and Guidelines**

Describe any goals, guidelines, principles, or priorities which dominate or embody the design of the system's software. Such goals might be:

* The KISS principle ("Keep it simple stupid!")
* Emphasis on speed versus memory use
* working, looking, or "feeling" like an existing product

For each such goal or guideline, unless it is implicitly obvious, describe the reason for its desirability. Feel free to state and describe each goal in its own subsubsection if you wish.

## **Development Methods**

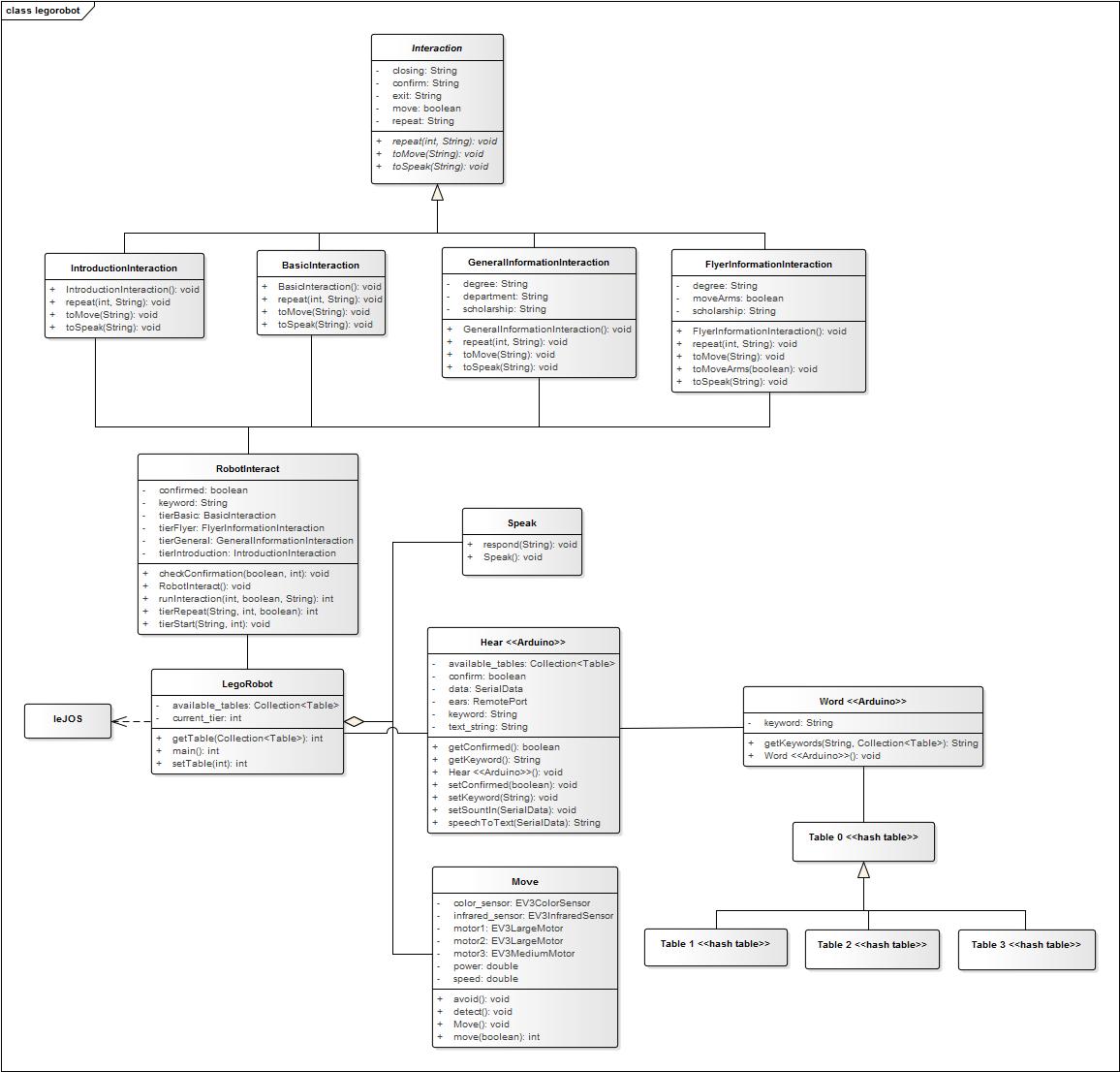
Briefly describe the method or approach used for this software design. If one or more formal/published methods were adopted or adapted, then include a reference to a more detailed description of these methods. If several methods were seriously considered, then each such method should be mentioned, along with a brief explanation of why all or part of it was used or not used.

# **Architectural Strategies (Lauren)**

The architectural design of LEA possesses portions of the strategy design pattern. This can be seen in the design of the interaction methods from the abstract interaction class. The Robot Interaction method sets what instance is invoked from which class, much like the strategy design pattern design. The design of LEA is based off the concept that new interactions can be added as this project progresses by simply adding another type of interaction without causing code reproduction. This layout provides low coupling and high cohesion that is essential for well-produced object oriented code. The strategy design pattern is the only one, thus far, that provides the behavioral needs enabling LEA to allow for the user to invoke different types of interactions.

# **System Architecture (Jess)**

LEA’s main function is to interact with students and guests. In order to accomplish this goal, LEA needs four major system components: a collection of interaction protocols and the abilities to speak, hear, and move around. LEA operates through moving independently of her environment. She will be able to detect objects in her environment and subsequently avoid these obstacles. This movement functions independently of the other system components; interacting is supported by the rest of the system.

*Figure 1: LEA’s Architecture* 

LEA contains modules for speaking and hearing. The speaking class is very basic and simply takes in a .wav file and sends it to the Lego Mindstorms EV3 speakers to be spoken. The listening module uses a collection of keywords in order to facilitate a conversation. When a specific keyword is heard, LEA has a predefined reaction based on the keyword. For example, LEA moves around her environment until the keyword “LEA” is spoken and understood by the Hear module, which is described in more detail in Section 4.1.1. This keyword ceases LEA’s movement in her environment and triggers an interaction module. This process continues in a tree-like manner to create a variety of conversation structures. The main Lego Robot module acts as a liaison between the hardware components of LEA (hearing, speaking, and moving) and the interaction logic for conversations.

The interaction logic is based off the Strategy design pattern. A base interface is designed with common functionalities that all interaction modules must follow. The separation of interactions was designed with future expansion in mind: a developer need only add a new interaction module in order to define additional behavior for LEA. This creates low coupling throughout the system and high cohesion in the interaction modules; only information needed for a particular interaction is contained in an interaction module. The interactions are created by the Robot Interact module, which is responsible for understanding which keyword was heard and returned from the Hear class and then spawning the proper interaction to continue the conversation.

## **Subsystem Architecture**

**4.1.1 Listening Ability**

The Hear class relies on a database connected to the Word class. The Word class contains a collection of keyword tiers. The tiers are defined in order to control the conversation tree. Depending on which interaction was spawned and where the conversation is currently, only one tier of keywords will be available for processing. This ensures that LEA does not get “confused” during an interaction and jump around the conversation tree; only the next keywords in the conversation tree will be available. For example, if the user is in the middle of interacting with LEA and they wish to be given a flyer, the phrase “LEA, please hand me a flyer” contains two keywords: “LEA” and “flyer”. Because “LEA” is a keyword only to start an interaction, it is no longer available as a recognizable keyword during a conversation. Therefore, only “flyer” remains and is the chosen keyword to continue the interaction.

These tiers follow a similar idea as the interactions. In order to add additional functionality, the developer need only add additional keywords in the proper tiers or add additional tiers to further extend the conversation tree.

**4.1.2 Robot Interact**

The Robot Interact class is responsible for the main conversation logic. The conversation logic was removed from the main module and segregated into a separate class in order to further encapsulate the conversation logic from the hardware logic, including the speaking, hearing, and moving, which are encapsulated and controlled by the main module. This module is responsible for confirming which keyword was heard and upon confirmation spawning the correct interaction based on which tier of the conversation tree the interaction is currently at, as described further in Section 6.2.

# **Policies and Tactics (La**uren)

In the development of LEA, an Arduino was used as the hearing mechanisms because it provided a more robust alternative to provide listening capabilities. The Arduino component is hardwired into LEA’s EV3 brick. All other components of LEA are developed from the various sensors provided by Legos and programmed through LEJOs.

As stated earlier, the interaction methods of LEA are based off the Strategy design pattern to ensure low coupling and high cohesion and therefore preventing unnecessary code reproduction.

**5.1 Code Repository**

GitHub will be used throughout this project as the repository for the source code. This tool is familiar to all members of this team and therefore is a viable tool to create and maintain code.

**5.2 Distribution of Duties**

Each team member will be responsible for various aspects of LEA’s code based off prior skills. For instance, the usage of the outside Arduino component will be developed solely by Stacey because she is the only one with the prior skills in this category.

# **Detailed System Design (Jess)**

**6.1 Interactions**

**6.1.1 Classification**

The interactions are a module of classes following the Strategy design pattern.

**6.1.2 Definition**

The purpose of the interaction classes is to encapsulate and define a variety of interactions that can be performed by LEA.

**6.1.3 Responsibilities**

The interactions classes are based off the Interaction interface. Each interaction class is responsible for defining a particular interaction between LEA and a user. Each interaction contains the responses available to LEA for that particular interaction. This creates a plug-and-play kind of system for defining LEA’s interaction abilities. A developer can easily add or remove interaction classes to change LEA’s conversation tree. In the context of the system design, the interaction module provides only the proper phrases and responses for each stage of the conversation tree. These classes do not contain any logic.

**6.1.4 Constraints**

An interaction class must be completely known before adding it to the system. All phrases and responses must be predefined. It is also assumed that if a new interaction class is added it has the necessary support added in the Robot Interact class and the proper keywords added in the Word class tiers.

**6.1.5 Composition**

There are no major subcomponents to this module. The module simply consists of a collection of interaction classes that operate independently of each other.

**6.1.6 Uses/Interactions**

Only one interaction instance can exist at one time in the system. The Robot Interact class is responsible for instantiating interactions and keeping track of the conversation tree. No other parts of the system interact directly with the interactions.

**6.1.7 Resources**

This module does not utilize any outside resources.

**6.1.8 Processing**

The interaction classes do not contain any program logic. Each class exists to provide their specific implementation of the conversation tree. The Robot Interact class utilizes the methods contained in each interaction class to create a conversation with the user.

**6.1.9 Interface/Exports**

This component mainly contains the .wav files needed to respond to a user in each particular interaction. The repeat and closing menu responses can also be altered and defined separately depending on the type of interaction.

**6.1.10 Detailed Subsystem Design**

Each interaction class contains a method to send a predefined phrase to the Speak class or a predefined value to the Move class if the interaction requires movement by LEA. Closing, confirmation, exit, and repeat messages are also defined in each class and provide unique phrases and responses for each interaction as .wav files. Each class will contain all the information needed for the interaction to be completed as desired by the programmer; it is the responsibility of the Robot Interact class to manage the interaction and provide interaction logic.

**6.2 Robot Interact**

**6.2.1 Classification**

Robot Interact is a class.

**6.2.2 Definition**

The purpose of Robot Interact is to encapsulate and define the behavior and logic for all interactions that can be performed by LEA and to maintain the state of the conversation tree.

**6.2.3 Responsibilities**

Robot Interact is one of the most important components in the system. It is responsible for validating keywords heard from the Hear class, maintaining the current position in the conversation tree, and determining which interaction to start to continue the current conversation. Robot Interact can be thought of as LEA’s brain and performs all functions related to interacting with the user, as well as commanding other system components such as Speak and Move.

**6.2.4 Constraints**

It is assumed that all interaction components interact with Robot Interact and provide it with the proper information it needs to maintain the conversation tree and current interaction behaviors.

**6.2.5 Composition**

There are no subcomponents to this class.

**6.2.6 Uses/Interactions**

The Lego Robot main class uses Robot Interact as a liaison between the hardware and interaction capabilities. Robot Interact instantiates and executes the interaction classes to create a conversation with the user. The information and responses from these classes are then fed through Lego Robot to the proper system component to create speech, movement, and hearing capabilities to facilitate conversations. This class interacts closely with Lego Robot and Hear to maintain system state.

**6.2.7 Resources**

This class does not utilize any outside resources.

**6.2.8 Processing**

Robot Interact contains a main method that determines which part of the conversation tree to instantiate. It uses the keyword determined by Hear and uses a combination of if-else and switch structures to determine if LEA needs to start an interaction (new tier of the conversation tree), repeat a question or statement, or exit the conversation tree and reset. Robot Interact has the ability to exit a conversation if no response from the user is heard or understood, allowing LEA to reset and avoid deadlocks if a user walks away without ending the conversation as expected.

**6.2.9 Interface/Exports**

This class associates the interaction classes to the rest of the system and maintains the current keyword being processed and if it was confirmed.

**6.2.10 Detailed Subsystem Design**

This class starts in the confirmation method, which allows Hear to confirm from the user which option in the conversation tree they are selecting. This confirmation happens with every new keyword in order to ensure the conversation is proceeding as the user desired. After confirmation, the keyword is used in a switch structure to determine which interaction needs to be instantiated to continue the conversation. A method is used to start the desired interaction and follow the logic necessary to complete the particular interaction. Upon completing an interaction, the interaction instance is destroyed and the Robot Interact state is reset to allow for a new confirmation and continuation of the conversation tree.

**6.3 Listening Ability**

**6.3.1 Classification**

The listening ability is a module containing the Hear and Word classes as well as the keyword tier tables, all contained within Arduino programming.

**6.3.2 Definition**

The purpose of this module is to provide LEA with the ability to listen to users and process keywords out of their phrases in order to engage in conversations.

**6.3.3 Responsibilities**

The listening module is responsible for determining predefined keywords from users’ speech. The module must listen at all times, process the incoming speech, and determine if there is a match for a keyword in the current conversation tree tier, as defined by the tier tables.

**6.3.4 Constraints**

Upon initial start-up, the conversation tree starts at the very top, with only the keyword “LEA” being recognizable. It is assumed that the user will speak clearly and that all predefined keywords have a matching interaction protocol.

**6.3.5 Composition**

The Hear class contains the main logic and hardware access to the microphone. It makes calls to the Word class when there is incoming speech data that needs to be processed. The Word class processes the data and searches the current tier table for a matching keyword. The table structure consists of a main table with keywords that are available during all sections of the conversation tree (e.g. “repeat”, “exit”, or “unknown”) and subtables that extend from the main table. The subtables define keywords that are only available for the current conversation tree tier, a mechanism that should allow for LEA to avoid jumping around in the conversation tree. The subtables follow a similar idea as the interaction classes, allowing for a plug-and-play system where new keywords can be added into existing tables or by adding a new table.

**6.3.6 Uses/Interactions**

Lego Robot associates with the listening module through a hard serial port connecting the Arduino panel to the Lego Mindstorms EV3 brick. Lego Robot acts as a liaison between the listening module and the Robot Interact interaction logic.

**6.3.7 Resources**

The listening module relies on an Arduino and microphone to provide the listening capabilities. This is then connected to the EV3 brick.

**6.3.8 Processing**

The Hear class will use speech to text software capabilities to turn spoken language into text within the system. This text can then be searched for keywords. Keywords of the same tier may be prioritized to avoid cases of multiple keywords in one statement leading to deadlock. The determined keyword will then be returned and confirmed by Robot Interact before an interaction can take place.

**6.3.9 Interface/Exports**

The listening interface contains a collection of tables that are maintained and used to determine which tier of the conversation tree is currently being accessed. It also contains a microphone reference for listening to spoken language.

**6.3.10 Detailed Subsystem Design**

This class starts with access to the microphone, which listens at all times and feeds the data to a buffer. This buffer is then processed by the Word class until a keyword is found. The keyword is confirmed and then sent through Lego Robot to Robot Interact to be processed. After processing, a conversation is initiated or a reset is performed.

# **Glossary**

LEA - *Lightweight Educational Assistant*

# **Bibliography**