Chapter 6

A GENERIC BEHAVIOR MANAGEMENT ARCHITECTURE FOR THE LANGUAGE FACULTY OF AN AGENT

The architecture of a system is the one which gives the technical blueprint of the system in terms of the required components, their organization and interactions. These aspects have to be identified and defined before the system development begins. In order to develop an agent with the specified language ability, its conceptual and internal state models, which are existing only theoretically as yet, have to be transformed into an architecture.

The architecture that is to be defined should accommodate the solutions that have been provided with respect to the conceptual model and the paradigm. For achieving this, the required architectural strategy which embodies the architectural knowledge about the solution approach to be used has to be identified. Following this, the required components, their organization and their interactions are to be identified. This process should align with the identified architectural strategy. These details have to be explored till the required level of abstraction of the architecture is reached. Finally, the architecture has to be evaluated for proving its validity.

Thus, the objectives of this chapter are the following:

- Devising the abstract architecture of the language faculty by identifying the requirements of the architecture, architectural strategy and major components
- Formulating the detailed architecture to the required level of abstraction by identifying the required components, their organization and interactions and
- Evaluating the architecture to justify its validity.
6.1 Abstract Architecture

In order to define the abstract architecture of the language faculty, the following three activities are carried out:

i. Identifying the requirements of the architecture

ii. Determining the architectural Strategy

iii. Identifying the abstract components and organizing them.

i. In preliminary to defining the architectural model of language faculty, the various requirements to be fulfilled by the architecture are described here. They are:

- Accommodating the conceptual model
- Realizing the Belief Task Behavior (BTB) based internal state of the language faculty
- Fulfilling the quality attributes of usability, modularity, openness, dynamicity, performance, scalability, maintainability, reusability and simplicity in the architecture.

The first two requirements follow from the previous two chapters and the qualitative attribute requirements follow from the hypothesis II given in problem statement. These attributes were identified to be essential to overcome the design limitations in the language ability of existing agents.

ii. In order to define the architecture that fulfills the above requirements, the required architectural strategy has to be identified first. This would form the basis of the architecture that would help to resolve the arising issues and also guide the development of the architecture. The discussion below describes the same.

The previous chapters described that the language faculty is to be realized as a combination of language management autonomy and language behavior autonomy. The management autonomy is constituted by four different autonomies. These autonomies correspond to the various management tasks that the language faculty has to perform, to manage the language behaviors in the various languages. These tasks could be generically visualized as behavior
management tasks. Hence, the architectural strategy should help to lay emphasis on behavior management with modular distribution of the management functions and organize the language behaviors under these management functions in such a way that the language behaviors are best managed by these management tasks.

This would shift the focus of the agent architecture from the conventional behavior based architectures like reactive (Brooks, 1986), BDI (Rao and Georgeff, 1991) and hybrid (Muller et al, 1995), (Wooldridge, 1999), to that of a behavior management based architecture where the required behavior management task takes the control first, followed by the behavior. This would also help the BTB paradigm to naturally fit into the architecture. Thus, the architectural strategy for defining the abstract architecture would be to formulate a Behavior Management based architecture.

iii. The architectural strategy requires that, based on the percept received, the behavior management takes control which is followed by the behavior. Hence, behavior management and behavior contribute for the two major components. Additionally, an interface component is required to interface with the environment. In order to facilitate the control flow as required by the architectural strategy, the three components are to be arranged in a three layered organization as shown in Figure 6.1. The resulting abstract architecture would consist of three layers namely, the interface layer, behavior management layer and the behavior layer.

![Figure 6.1: Abstract Architecture of the Language Faculty](image-url)
These layers communicate with each other by using messages. Whereas the existing agents are composed of only the interface and behavior layers, the behavior management layer is the new layer which is introduced in this work.

A more generic introspection and abstraction of the above architecture would help to discern that behavior management is not only required for the language faculty, but for any functional task of an agent as has been illustrated in the travel planning agent and ontology services agent described in the previous chapter. Hence, the four language ability management tasks could be generically visualized as:

- Competence Behavior Management
- Knowledge Behavior Management
- Configuration Behavior Management
- New Behavior Acquisition Management

The existing agent architectures focus only on designing the functional behavior of an agent. On the other hand, realizing a more refined form of an agent requires that the functional behavior and its architecture are supported by the required management functions. This would help to elevate the functional component of an agent into a faculty that possesses the above described management functions as well as the behavior functions. Thus, the Behavior Management architecture would be applicable to any function performed by the agent.

6.2 Detailed Architecture of the Language Faculty

The detailed architecture of the language faculty is explored at two levels of abstraction viz.

- Macro level
- Micro level.

The Macro level describes the overall architecture of the language faculty and the micro level delves into the internal architecture of each of the three layers of the language faculty.
6.2.1 Macro Level Architecture

The description of the macro architecture is expressed in terms of the following:

- Identification of the Components
- Organization of the Components
- Interaction of the Components.

6.2.1.1 Identification of the Components

This section describes about the components that make up the interface layer, behavior management layer and the behavior layer.

An agent with language ability would possess two levels of interfaces. One external interface that consists of the components for authenticating with the agent and changing the language of interaction manually and the other internal interface that helps to perceive and respond to the language-based input. The second level interface is explained in the discussions below.

For the Behavior Management layer, the four tasks of the behavior management layer are considered and the components that are required to accomplish these tasks are given in Table 6.1.

<table>
<thead>
<tr>
<th>Task</th>
<th>Components Identified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Competence Behavior Management</td>
<td>Interface Behavior Management Component</td>
</tr>
<tr>
<td></td>
<td>Function Behavior Management Component</td>
</tr>
<tr>
<td>Knowledge Behavior Management</td>
<td>Knowledge Behavior Management Component</td>
</tr>
<tr>
<td>Configuration Behavior Management</td>
<td>Task functionality to be attributed within the other identified management components.</td>
</tr>
<tr>
<td>New Behavior Acquisition Management</td>
<td>New Behavior Acquisition Management Component</td>
</tr>
</tbody>
</table>
components. The interface management helps to manage the perceive and response behaviors in every language supported in the second level interface. The function management helps to manage the CNLI rendering behaviors of every language. The configuration behavior management is proposed to be accomplished by the management components themselves as will be described in the subsequent micro level architecture.

The behavior layer is composed of behavior components corresponding to each of the management components. The component level architecture of the language faculty is given Figure 6.2.

![Architecture of the language faculty of an agent](image)

**Figure 6.2: Architecture of the language faculty of an agent**

6.2.1.2 Organization of the Components

The overall organization of the language faculty is a vertically layered organization with two pass control as shown in Figure 6.3. The two pass control signifies that the information flows up the architecture in the first pass and the control flows down in
the second pass. That is, the user's natural language request flows up to the functional ability layer through the lower language faculty layer and the action or result is conveyed again through this language faculty layer.

Figure 6.3: Vertical Organization of an agent with language ability

Integrating the three layered organization proposed for language faculty and the vertical organization of an agent with language ability, the overall organization is as given in Figure 6.4.

6.2.1.3 Interaction between the components

At the highest level of abstraction, the interaction happens between the interface layer, behavior management layer and the behavior layer. But, the overall functionality of the language faculty is achieved through the interactions between the components of the behavior management layer. This is because the components of the various behavior layers do not communicate directly. All their interactions happen only through the behavior management layer. Hence, the interactions that happen between the behavior management layer components alone are explained.

The interactions between the components and the purpose of their interactions with respect to the language faculty are described in Table 6.2. Since the knowledge
behavior management component interacts with all the other three management components, its interactions are not given separately in the table.

![Diagram of Language Faculty Components]

**Figure 6.4: Overall organization of Language Faculty Components**

**Table 6.2: Components of the Behavior Management Layer and their Interactions**

<table>
<thead>
<tr>
<th>Component</th>
<th>Name of interacting component</th>
<th>Purpose of interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Behavior Management Component</td>
<td>Function Behavior Management component</td>
<td>To convey the percept received and receive the response of the function behaviors.</td>
</tr>
<tr>
<td>Knowledge Behavior Management Component</td>
<td>Knowledge Behavior Management component</td>
<td>To obtain knowledge resources for interface behaviors.</td>
</tr>
<tr>
<td>Function Behavior Management Component</td>
<td>Interface Behavior Management component</td>
<td>To obtain percept from and convey response to the environment.</td>
</tr>
</tbody>
</table>
6.2.2 Micro Level Architecture

The micro level architecture explores into the internal details of the components of the three layers. The external interface component has a trivial function, and hence its architecture is not explored into at the micro level. The architecture of the behavior layer components is already available in the literature as described below.

Architectures for the interface behaviors are contributed by the field of human computer interaction (Clerckx et al, 2006), (Stephanidis et al, 1997), (Hefley and Murray, 1993).

Architectures for language function behaviors pertaining to NLP, NLG and dialoguing (Crocker et al, 1999) are contributed by the field of computational linguistics.

The behaviors corresponding to the language knowledge services are language knowledge representation dependent and corresponding architectures that help to store and retrieve knowledge in this representation are available (Harmelen, 2007), (Russel and Norvig, 1995).

New Language acquisition behavior helps to acquire new languages. Language acquisition could be achieved cognitively or functionally. In the cognitive approach, language learning procedures (ws10), (ws11) that make use of machine learning algorithms are used (Hu and Atwell, 2005). In the case of functional approach, knowledge acquisition functions that help to acquire the language knowledge automatically or semi-automatically from a knowledge engineer (Lavrac and Mozetic, 1989). Architectures for both of these language acquisition behaviors are available.
Thus, there are several architectures available at the behavioral level. But, the behavior management has to focus on managing the behaviors, for which suitable architectures are yet to be evolved. Thus, there is a need for formulating the architecture for behavior management of an agent. The architecture for behavior management layer and the internal architecture of the management functions have to be explored.

For proposing the micro level architecture of the behavior management layer components, the characteristic of these management components need to be considered first. This is explained below.

6.2.2.1 Characteristics of the Behavior Management Layer components

The four components corresponding to behavior management layer help the agent to manage its interface, function and knowledge in all the languages supported and also acquire knowledge pertaining to all the above three in a new language. These management tasks actually help the agent to self-manage it. Usually, a self-management system is expected to exhibit the following properties:

- **Self-organization** – self organizing systems must adapt dynamically to environmental changes by re-organizing themselves.
- **Self-adaptation** – self adaptive software is one which modifies its behavior in response to changes in its operating environment. A system is open adaptive if new behaviors could be introduced during run time.
- **Self-configuration** – can dynamically adapt to changes in the environment, where the changes could include the deployment of new components, or removal of existing ones.
- **Self-healing** – can discover, diagnose and react to disruptions automatically. They can detect system malfunctions and initiate corrective actions.
- **Self-optimization** – can monitor and tune resources automatically to meet end-user or business needs.
- **Self-protecting** – can anticipate, detect, identify and protect against threats. They can take corrective actions to make themselves less vulnerable.
The above properties have been borrowed from the two contexts that discuss about self-management namely, software architecture (Kramer and Magee, 2007) and autonomic computing (Serugendo et al, 2006), (ws12). The latter three properties, though are relevant with respect to the self-management, are beyond the scope of this work. Hence, the former three properties are considered and worked upon to formulate the micro level architecture.

Since the behavior management components should help to provide for self-management, the design issues for each of the self-management properties – self-organization, self-adaptation and self-configuration are considered individually and suitable solutions are proposed. Using these proposed solutions, the micro-level architecture is built progressively for each of the four behavior management components. These are explained below.

6.2.2.2 Design issues and Solutions for Self-Organization, Self-Adaptation and Self-Configuration

This section describes about the design issues and solutions that are being proposed in this work for self-organization, self-adaptation and self-configuration. This is because all the three properties are required to be exhibited by every behavior management component. Using these solutions, the architectures for the four behavior management components are built.

Self-Organization Issues and Solutions

Self-organization relates to the capacity of the system to spontaneously produce a new organization in case of environmental changes. That is, components should be able to dynamically bind together under user requests or for pursuing their own goals. This requires that a component listens to events or percepts in the environment and carries out the dynamic binding. The various design alternatives in incorporating the listener and carrying out the dynamic binding are as follows.

Alternative 1

Listeners could be put in each of the behavior components and the behavior component that recognizes its event or percept gets activated.
Limitations

- Every behavior component should have its own listeners.
- The percept should be broadcast to all behaviors resulting in communication overhead.
- Communication failure would result in the required behavior not being activated.
- Not compatible with the macro architecture described above as there is no role for behavior management in behavior initiation which should have been the case.

Alternative 2

The second alternative is to have a listener in the behavior management component instead of the behavior components. Based on the percept that is received, the behavior management component decides which of the behavior component to activate thereby self-organizing itself for providing the corresponding behavior.

Limitations

- Though it is compatible with the above described behavior management architecture, yet resources for the two components (two processes) are required—the behavior management and the behavior. This is an overhead on the system.

Alternative 3

This alternative is closely aligned with that of the above alternative, with the listener incorporated with behavior management. The difference is that the behavior management is conceived to be composed of manager and behaviors roles. The behavior manager role is active by default and the listener waits for the relevant percept that any of the behaviors under its control could accomplish. When it receives the relevant percept it relinquishes its behavior manager role and takes on the required behavior role. That is, the dynamic binding that is very much required of self-organization is accomplished in the most effective manner. After completion of the behavior role, it resumes the behavior management role. This is depicted in Figure 6.5.
Among the three alternatives, the third alternative is the one chosen for achieving self-organization. The architecture of the management components is proposed using this alternative only. This helps to achieve the behavior configuration management in every component by the dynamic binding to the required language behavior role.

**Self-Adaptation Issues and Solution**

The functional behavior of the agent should adapt to the user so as to provide the preferred behavior. In order to provide for this, both the functional behavior as well as the knowledge required to perform the functional behavior should be configured accordingly. There are two ways of achieving this adaptation as follows:

- Every behavior component has individual observers to observe the user interactions and deduce the behavior preferred by the user.
- A separate component tries to observe the user and deduces the adaptation parameters for the behavior. It intimates these parameters of adaptation to the other components so that they can adapt themselves accordingly.

In the former approach, which is proposed in (Kramer and Magee, 2007), the limitation is that there is an overhead of observers in every behavior component. Also, it does not comply with the above proposed macro architecture. There is also possibility that there is inconsistency between the deduced values of the adaptation parameters between the various observers. In the latter approach, which is suggested in this thesis, there is a single observing component, which deduces the parameters of adaptation and intimates the other components that require it. This approach helps to overcome the above limitations.
Since, the user has to be observed so as to deduce his behavior preferences, it requires that this observing component is part of the interface. Hence, adaptation is realized as part of Interface Behavior Management component. These adaptation values are conveyed to the knowledge and competence management components which adopts the corresponding language behavior role in order to provide service to the user in the corresponding language.

**Self-Configuration Issues and Solution**

A self-configuration architecture will be able to acquire new services and make them available for use in a seamless manner. The two alternative ways of achieving this is as follows:

- Every behavior management component acquires the required competence and knowledge and deploys it.
- A separate behavior acquisition management component acquires the required competence and knowledge for all the three behavior management components which is deployed by them individually.

Both of the alternatives could be supported by the proposed micro level architecture. But, the latter alternative is more appropriate as it follows from the macro level architecture given in Figure 6.2. Another compelling reason for deciding this alternative is because the new behavior acquisition function requires a combined view of both the functional domain ontology and language ontology. Since, these ontologies are separated between function behavior management and knowledge behavior management respectively, the overall view of domain ontology and functional domain ontology is not possible, if the new behavior acquisition function is shared between the three management components, which is the case in the first alternative. Hence, the latter approach is found to be appropriate in all aspects. Thereby, the behavior acquisition management helps to acquire the interface behavior, function behavior and knowledge for the newly acquired language behavior.

While mapping the above solutions to the architecture of the behavior management components, only the self-organization property is implemented individually in the architectures of the four management components by using the dynamic binding
between the manager and behavior roles. Hence, the four components share the same basic architecture. But, for achieving self-adaptation and self-configuration properties, one behavior management component accommodates the required sub-component(s) in its architecture. It shares the result of its work with other management components, thereby enabling them also to exhibit the self-adaptation and self-configuration properties. That is, the interface behavior management component alone has sub-components for achieving self-adaptation and the new behavior acquisition management component alone has provision for self-configuration in their architecture. They share the result of their work with the other behavior management components enabling them also to exhibit the corresponding property. The subsequent discussion describes about the same.

6.2.2.3 Architecture of the Function Behavior and Knowledge Behavior Management Components

Using the above described solutions for self-organization and the definition of the Task Component in the BTB paradigm described in the previous chapter, the basic architecture of a behavior manager component is given in Figure 6.6. This architecture is suitable for supporting function behavior and knowledge behavior management components.

![Figure 6.6: Internal Architecture of the Behavior Management](image)

The incoming percept is received by the manager role interface. This component corresponds to the listener component given in Figure 6.5. This percept is used to update the management task context. The task context maintenance component is
responsible for maintaining the task context regarding the corresponding management task. Metadata about every behavior role helps the role management to keep track of the various behavior roles it maintains, under what conditions they should be activated, the procedures for their activation etc. Both these maintenance functions include typical functions of adding and updating the corresponding data which it maintains. The behavior roles management component determines which of the behavior roles to assume based on the current task context and activates the required behavior role. The functions which the behavior role management should support are as follows:

- Dynamic Role binding
- Role inclusion
- Role deletion
- Role Management level inference.

Dynamic role binding helps the manager role to bind to the required behavior role. This helps to enforce the configuration management autonomy, whereby, the manager configures to the appropriate behavior role when required. Role inclusion is required to include a new role in the set of roles being supported. This is achieved by updating the role metadata and the task context. Role management level inference helps to perform a management level inference of the behavior roles like that of which of the behavior roles is very reliable, dependable, or lengthiest etc. This function is required when the other aspects of self-management like that of self-healing, self-protection etc. have to be supported. The ontology reference corresponds to function domain ontology in the case of function behavior management and language ontology in the case of knowledge behavior management.

Now, the solution for the design issues of other two properties of self-adaptation and self-configuration are considered and changes to the above architecture, are made to result in the architecture of the interface behavior management and new behavior acquisition management.

6.2.2.4 Architecture of the Interface Behavior Management Component

The architecture of this component is similar to that given in Figure 6.6 except that the behavior role management has an additional adaptation and intimation component
as given in Figure 6.7. This new sub-component helps to observe the user and model his preferences and convey the same to other components in order to provide for an adaptive behavior. Since this component requires to control only the interface behavior roles, it does not require any ontology reference as in the case of the function behavior and knowledge behavior management components. Hence, this is not given in the architecture.

![Figure 6.7: Architecture of the Interface Behavior Management Component](image)

**6.2.2.5 Architecture of the New Behavior Acquisition Management Component**

The same architecture given in Figure 6.6 is used with the only difference that the ontology reference has both the function domain ontology and generic language ontology. Both of them are required for acquiring a new language behavior. The acquired behavior is deployed in the interface, function and knowledge management components using their role inclusion function. The architecture is given in Figure 6.8.

Thus, the architecture of the behavior management layer at two levels of abstraction has been described. Table 6.3 summarizes how the three self-management properties are fulfilled in the architecture.
New Behavior Acquisition Manager Role

Table 6.3: Properties of Self-Management and their fulfillment by the language faculty

<table>
<thead>
<tr>
<th>Property</th>
<th>Fulfillment of property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Organization</td>
<td>the interface, function and knowledge management components are dynamically bound with the required language behaviors in order to provide the interaction in the required or preferred language.</td>
</tr>
<tr>
<td>Self-Adaptation</td>
<td>Language preferences of the user are deduced by the adaptation sub component in the interface behavior management component by observing the user’s language use and the working environment in the corresponding language is provided automatically.</td>
</tr>
<tr>
<td></td>
<td>The adaptation is also open adaptive as new language behaviors could be automatically introduced without user support by the new behavior acquisition component.</td>
</tr>
<tr>
<td>Self-Configuration</td>
<td>New Behavior acquisition component acquires new language behaviors. It deploys these behaviors in the language faculty without user involvement.</td>
</tr>
</tbody>
</table>
6.3 Proposed Behavior Management Architecture Vs Existing Self-Management Architecture

The described behavior management architecture provides for self-management architecture of agents. In the existing literature, there is an abstract and generic three layered self-management architecture proposed by Kramer and Magee (Kramer and Magee, 2007). They have identified certain issues that should be addressed by each of the three layers for which solutions are yet to be found. Since the behavior management architecture provides for solutions for these issues, its architecture is compared with the abstract architecture to evince its advantages.

Figure 6.9 contrasts the proposed behavior management architecture for agents with that of the existing self-management architecture proposed by Kramer and Magee (Kramer and Magee, 2007). The behavior layer corresponds to the component control layer. The behavior layer consists of the behaviors corresponding to each of the management functions. The behavior management layer corresponds to the change management layer. The behavior management layer has all the management components clearly identified. The internal architecture of these components is also described above. The distribution of the management functions follow from the various aspects of an agent functionality management namely the interface, competence, knowledge, and augmentation of the three aspects using acquisition.

Table 6.4 enlists the issues that are specified to be addressed in the abstract self-management architecture and the solutions provided in the behavior management architecture.

Thus, the behavior management architecture provides the solutions for the various issues identified in the abstract self-management architecture resulting in a self-management architecture for the language faculty of agents.
Figure 6.9: Comparison of abstract Self-Management Architecture against the proposed Behavior Management Architecture for language faculty of an agent
Table 6.4: Issues in the Layers of Self-Management and corresponding solutions in proposed Behavior Management Architecture

<table>
<thead>
<tr>
<th>Name of Layer in Self-Management Architecture</th>
<th>Issue</th>
<th>Solution in Behavior Management Architecture for Language Faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component control</td>
<td>Preserving safe application operation during change.</td>
<td>Until the components are self-organized appropriately, the language faculty is in the passive state.</td>
</tr>
<tr>
<td>Change management</td>
<td>Distribution and decentralization of change management functions.</td>
<td>Change management functions are distributed in terms of changes in interface, function and knowledge corresponding to the management components.</td>
</tr>
<tr>
<td>Goal Management</td>
<td>Precise specification of application goals and system goals concerned with self-management.</td>
<td>The language faculty is able to acquire new languages and provide its services in that language. This is taken care of by the new behavior acquisition management component.</td>
</tr>
</tbody>
</table>

6.4 Evaluation of Architecture

This section provides an overview of the need for the evaluation of architecture, the various quality attributes used for evaluating architectures, the three methodologies available for evaluating architectures and finally converges on the appropriate method which is suitable for evaluating the behavior management architecture of the language faculty of agents. Finally, the evaluation of the architecture using the chosen architecture evaluation method is described.

Architecture is the product of the early design phase, and its effect on the system and the project is profound. An unsuitable architecture will precipitate disaster on a project. Performance goals will not be met. The right functionality is not available and the system is too hard to change. Architecture evaluation is a cheap way to avoid disaster.
6.4.1 Need for Architectural Evaluation (Clements et al, 2002)

Architectural evaluation produces answers to two kinds of questions:

- Is the architecture suitable for the system for which it was designed?
  Suitability is relevant in the context of specific goals for the architecture and the system it spawns. Hence, the overarching part of an architecture evaluation is to capture and prioritize specific goals that the architecture must meet in order to be considered.

  An architecture is said to be suitable if it meets two criteria
  
  - The system that results from it will meet its quality goals. For example, the system will run predictably and fast enough to meet its performance (timing) requirements. It will be modifiable in planned ways. It will meet its security constraints. It will provide the required behavioral function etc.
  
  - The system can be built using the resources at hand: the staff, the budget, the legacy software (if any), and the time allotted before delivery. That is, the architecture is buildable.

- Which of two or more competing architectures is the most suitable one for the system at hand?
  Architecture evaluation can be applied to a single architecture or to a group of competing architectures. In the latter case, it can reveal the strengths and weaknesses of each one.

6.4.2 Method for Evaluating the Behavior Management Architecture

The following are the three methods are available for architecture evaluation (Clements et al, 2002):

- ATAM: Architecture Tradeoff Analysis Method
  This method gets its name because it not only reveals how well an architecture satisfies particular quality goals but it also provides insight into how those quality goals interact with each other – how they tradeoff against each other.
• SAAM: Software Architecture Analysis Method
  This is one of the most widely used architecture analysis method. It helps to show that an architecture actually fulfills the quality attributes claimed to be possessed by the architecture. It also helps to evaluate the functional coverage of the architecture. It is the predecessor of the ATAM method which inherits many of its steps of evaluation.

• ARID: Active Reviews for Intermediate Designs
  This evaluation method is used to evaluate a partial design in its infancy. This is in contrast to the ATAM and SAAM methodologies which are comprehensive methods for evaluating a complete architecture. This method helps to determine whether a design being proposed is suitable from the point of view of other parts of the architecture that will be required to use it.

Each of the three methods available for architectural evaluation is unique in nature and specifically applicable for evaluating certain aspects of an architecture, during specific project stages. From a comparison of the three architecture evaluation methods, the SAAM method is found to be best suited for evaluating the behavior management architecture of language faculty as it helps to fulfill the requirements of evaluation. The subsequent section describes the evaluation of the behavior management architecture using the SAAM methodology.

6.4.3 Evaluation of Behavior Management Architecture

In this section, an overview of the steps involved in the SAAM methodology and the application of the SAAM methodology to evaluate the above architecture is described.

6.4.3.1 SAAM Methodology

This section gives an overview of the steps involved in the SAAM Methodology. The SAAM asks the system's various stakeholders to enumerate a set of scenarios that represent the known or likely changes that the system will undergo in the future. These scenarios are scrutinized, prioritized and mapped onto a representation of the architecture. The steps of the SAAM methodology are described below.
Develop Scenarios.
A scenario is a short statement describing an interaction of one of the stakeholders with the system. In developing the scenarios, it is crucial to capture all of the major uses of a system, users of a system, anticipated changes to the system and qualities that a system must satisfy now and in the foreseeable future.

Describe the Architecture
The candidate architecture or architectures should be described in an architectural notation that is well understood by the parties involved in the analysis. These architectural descriptions must indicate the system's computation and data components as well as all relevant connections.

Classify and Prioritize the Scenarios
A scenario in the SAAM is a brief description of some anticipated or desired use of a system. SAAM classifies the scenarios as direct or indirect scenarios. Direct scenarios are those that are satisfied by the architecture through the execution. They correspond to requirements previously addressed in the design process.

An indirect scenario is one that requires a modification to the architecture to be satisfied. Indirect scenarios are central to the measurement of the degree to which an architecture can accommodate evolutionary changes that are important to the stakeholders.

Individually Evaluate Indirect Scenarios:
Once a set of scenarios has been chosen for consideration, these scenarios are mapped onto the architectural description. In the case of a direct scenario, the architect demonstrates how the scenario would be executed by the architecture. In the case of an indirect scenario, the architect describes how the architecture would need to be changed to accommodate the scenario. In addition, the cost of performing the change must be estimated. A modification to the architecture means that either a new component or connection is introduced or an existing component or connection requires a change in its specification.
Assess Scenario Interactions:
When two or more indirect scenarios require changes to a single component of an architecture, they are said to interact in that component. Scenario interaction is important to highlight for two reasons. First, it exposes the allocation of functionality to the product's design. The interaction of semantically unrelated scenarios explicitly shows which components are computing semantically unrelated functions.

Create the Overall Evaluation.
Finally, a weight must be assigned to each scenario in terms of its relative importance to the success of the system. The weighting can be used to determine an overall ranking if multiple architectures are being compared or if different architectural alternatives are being proposed for a single architecture. Assigning weights is a subjective process involving all of the stakeholders, but it should be done in a structured fashion.

6.4.3.2 Evaluation
Here, the evaluation of the Behavior Management architecture using the SAAM methodology is described. The activities carried out in every step of evaluation are given below.

Develop Scenarios.
The following are the scenarios which have been selected to be used for evaluation.
1. A known user logs in.
2. The user gives the specification of the delegated task.
3. The user's request is abstract and his intention of task could not be identified.
4. An existing language is removed.
5. The administrator wants to update the vocabulary.
6. A new language is required to be acquired.
7. The included language should be implemented.
8. The user changes the language of interaction.
9. The user wants a fast reply for his request.
10. The CNLI behavior in a particular language encounters a new word and tries to update the vocabulary.
11. The user gives a request containing words in more than one language.
12. Change the knowledge representation technique.
13. Change the procedure for CNLI.
15. Known user gives an abstract task specification.
16. Include provision for context-aware interaction.
17. Change the mode of interaction to multi-modal.
18. Add security features to the language faculty.
19. CNLI procedure in one language is not working.
20. Physically distribute the components of the language faculty.
21. Use the language faculty for interaction in micro devices.
22. Reuse language faculty for another task.

Describe the Architecture:
The architecture has already been described in the previous section and hence is not repeated here.

Classify and Prioritize the scenarios:
The scenarios identified above are classified into direct and indirect scenarios as given in Table 6.5. The scenarios are given in the order of their priorities.

Table 6.5: Direct and Indirect Scenarios to Evaluate Behavior Management Architecture

<table>
<thead>
<tr>
<th>Direct Scenario</th>
<th>Indirect Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A known user logs in.</td>
<td>1. The user gives a request containing words in more than one language</td>
</tr>
<tr>
<td>2. The user gives the specification of the delegated task.</td>
<td>2. Change the knowledge representation technique.</td>
</tr>
<tr>
<td>3. The user’s request is abstract and his intention of task could not be identified.</td>
<td>3. Change the procedure for CNLI.</td>
</tr>
<tr>
<td>4. An existing language is removed.</td>
<td>4. Include support for multiple-users.</td>
</tr>
<tr>
<td>5. The administrator wants to update the vocabulary.</td>
<td>5. Known user gives an abstract task specification.</td>
</tr>
<tr>
<td>6. A new language is required to be acquired.</td>
<td>6. Include provision for context-aware interaction.</td>
</tr>
<tr>
<td>7. The included language should be implemented.</td>
<td>7. Change the mode of interaction to multi-modal.</td>
</tr>
</tbody>
</table>
### Evaluating the Scenarios.

In this step, the direct and indirect scenarios are considered individually and how the architecture could provide for the same are described. This is given in Tables 6.6 and 6.7 The following are the abbreviations used in the tables.

IBM – Interface Behavior Management  
FBM – Function Behavior Management  
KBM – Knowledge Behavior Management  
NBAM - New Behavior Acquisition Management  
LFI – Language Faculty Interface

<table>
<thead>
<tr>
<th>Direct Scenario</th>
<th>Elements that help to achieve the scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>8. The user changes the language of interaction.</td>
<td>8. Add security features to the language faculty.</td>
</tr>
<tr>
<td>9. The user wants a fast reply for his request.</td>
<td>9. CNLI procedure in one language is not working.</td>
</tr>
<tr>
<td>10. The CNLI behavior in a particular language encounters a new word and tries to update the vocabulary.</td>
<td>10. Physically distribute the components of the language faculty.</td>
</tr>
<tr>
<td>11. Use the language faculty for interaction in micro devices.</td>
<td></td>
</tr>
<tr>
<td>12. Reuse language faculty for another task.</td>
<td></td>
</tr>
</tbody>
</table>

**Table 6.6: Evaluation of Direct Scenarios**

<table>
<thead>
<tr>
<th>Direct Scenario</th>
<th>Elements that help to achieve the scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A known user logs in.</td>
<td>- The IBM which always observes the user's language use in fields and stores his preferred language and updates it regularly  &lt;br&gt;  - It sends this preferred language detail to the FBM and KBM which dynamically assume the corresponding language behavior role.</td>
</tr>
<tr>
<td>2. The user gives the specification of the delegated task.</td>
<td>- At the time of receiving a task request, the FBM has taken on the required CNLI language behavior role and KBM is ready to provide knowledge services to the corresponding language.  &lt;br&gt;  - To process the request, the CNLI behavior role requests the knowledge from the KBM.</td>
</tr>
<tr>
<td>3. The user's request is abstract and his intention of task could not be identified.</td>
<td>CNLI behavior role begins the collaboration process to infer the user's intentions.</td>
</tr>
<tr>
<td>4. The administrator wants to update the language knowledge.</td>
<td>This is achieved through KBM.</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>5. A new language is required to be acquired.</td>
<td>This is achieved through the NBAM.</td>
</tr>
</tbody>
</table>
| 6. The included language should be implemented. | - The NBAM sends the acquired competence to the FBM and knowledge to the KBM.  
- The corresponding role managers add the new behavior role using the role maintenance module. |
| 7. The user changes the language of interaction. | - The IBM makes note of the language change and sends the change to the FBM and KBM which are currently in a particular language behavior role.  
- They disable that role and resume the corresponding manager role from which they dynamically assume the required behavior role. |
| 8. The user wants a fast reply for his request. | - This is possible because though the FBM and KBM are involved, they are in the corresponding behavior roles at the time of receiving the request  
- Hence, the manager role overhead is not there. |
| 9. The CNLI behavior in a particular language encounters a new word and tries to update the vocabulary. | - This is not allowed because, FBM and KBM are cohesive components that can communicate only by messages.  
- So a CNLI behavior of FBM cannot update the knowledge and only the KBM can do it. |
| 10. Use different architectures for different language behavior roles. | Possible as they are implemented as separate roles and the role manager is not architecturally dependent on the behavior role. |

Table 6.7: Evaluation of Indirect Scenarios

<table>
<thead>
<tr>
<th>Indirect Scenario</th>
<th>Elements requiring change</th>
<th>Complexity involved in achieving the indirect scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The user gives a request containing words in more than one language.</td>
<td>A multilingual vocabulary construction module could be included in the KBM manager role, which could be made use of by the behavior roles of the FBM.</td>
<td>Simple</td>
</tr>
<tr>
<td>2. Change the knowledge representation technique.</td>
<td>- The knowledge servicing behavior roles of the KBM should be modified to store and retrieve knowledge in the new</td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Representation.</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>• The KBM manager role should include an interface module which converts the knowledge in the new representation to a mutually agreed format between the KBM and FBM.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Change the procedure for CNLI.</td>
<td>Only the behavior role procedure of the FBM has to be changed.</td>
</tr>
<tr>
<td>4</td>
<td>Change the architecture of the CNLI procedure.</td>
<td>Only the behavior role procedure of the FBM have to be changed.</td>
</tr>
</tbody>
</table>
| 5 | Include support for multiple-users concurrently                              | • Multiple users can be handled using separate threads. Every thread would contain the manager role bound to the behavior role in the language required or preferred by the user.  
    |                                                                 | • Thus every thread would contain the IBM, FBM and KBM bound to the required language behavior roles  
    |                                                                 | • The required change has to be done in the language faculty interface. | Medium |
| 6 | Include provision for context-aware interaction.                             | A context maintenance module could be included in the manager role of FBM. | Medium |
| 7 | Change the mode of interaction to multi-modal.                               | The IBM behavior roles should include modules for supporting multi-modal interaction. | Medium |
| 8 | Add security features to the language faculty.                              | A new security handling component should be included in the LFI to secure every in-bound and out-bound interaction. Thereby, self-protection could be achieved. | Simple |
| 9 | CNLI procedure in one language is not working.                               | Can acquire a new CNLI procedure for that language through the acquisition module. Thereby, self-healing could be achieved. | Simple |
| 10| Physically distribute the components of the language faculty.                | The behavior management modules communicate by sending messages. This communication module should be changed to include remote communication. | Simple |
| 11| Use the language faculty for interaction in micro devices.                   | The same architecture could be reused for developing language faculty for micro devices using the micro edition of the programming language. Thereby | Medium |
| 12. Reuse language faculty for another task. | Only the behavior roles of the FBM and KBM should be changed to process requests corresponding to that task domain and maintain the corresponding language respectively. | Simple |

**Assess Scenario Interactions:**

The following Table 6.8 gives the details of the scenario interactions. The row corresponding to scenario 11 is empty indicating that the entire language faculty has to be rewritten using the micro edition version of the programming language used, whereas, the architecture remains the same.

**Overall Evaluation**

The evaluation of the direct and indirect scenarios indicate that the architecture not only fulfills the current requirements but also, can support or has provision to support the future modifications that may be required. The scenario interactions in Table 6.8 indicate that the modifications require changes in mostly one component or two related components. Thereby, it is obvious that the architecture is composed of highly cohesive components with no coupling between them and also they are accommodative of any form of foreseen modifications. The cost incurred in doing the modifications is either 'simple' or 'medium', which indicates that the required changes could be easily achieved. Since, the architecture provides for the direct scenarios and also the indirect scenarios, it could be concluded that the architecture fulfills the present and foreseen requirements, contributing for a viable architecture.

**6.4.4 Fulfillment of Required Qualitative Properties**

In this section, the various qualitative attributes that were required to be fulfilled by the architecture are considered and how they are fulfilled by in the architecture of the language faculty is described in Table 6.9.
### Table 6.8: Scenario interactions

<table>
<thead>
<tr>
<th>Scenario No.</th>
<th>IBM</th>
<th>FBM</th>
<th>KBM</th>
<th>NBAM</th>
<th>LFI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mgr. Role</td>
<td>Behavior Role</td>
<td>Mgr. Role</td>
<td>Behavior Role</td>
<td>Mgr. Role</td>
</tr>
<tr>
<td>1.</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 6.9: Fulfillment of Quality Attribute Requirements by the Behavior Management Architecture

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Qualitative Attributes to be fulfilled by Architecture</th>
<th>Description</th>
<th>Aspects of language faculty architecture that helps to fulfill the requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Usability</td>
<td>Should help to provide for the language ability behaviors in the supported languages.</td>
<td>CNLI behaviors provided in all the supported languages.</td>
</tr>
<tr>
<td>2.</td>
<td>Modularity</td>
<td>Should provide for a language management architecture that helps to manage the multiple language competencies and knowledge modularly such that the competence and knowledge have least coupling between them.</td>
<td>Separate competence and knowledge management components that communicate by using messages.</td>
</tr>
</tbody>
</table>
### 6.5 Advantages of the Architecture

1. Introduces a new behavior management layer that helps the agent to self-manage its behavior in multiple dimensions.

2. The macro architecture as well as the micro architecture of the behavior management layer are not only applicable for the language faculty and its internals but it is a generic architecture applicable for realizing any functional faculty of an agent.
3. The architecture provides for a decentralized and distributed behavior management.

4. It explicits the components that are required in the behavior management layer in contrast to the existing abstract self-management architecture which is very abstract.

5. The behavior management components fulfill the self-management properties of self-organization, self-adaptation, self-configuration.

6. Conceptualizing behavior management and behaviors as roles helps each of these to be dynamically initiated.

7. Solutions to the issues identified in the abstract self-management architecture have been provided.

6.6 Summary

The conceptual model of the language faculty and the defined BTB internal state paradigm form the basis for constructing the architecture of the language faculty. The architecture design process is initiated by consolidating the requirements and the identification of an architectural strategy that would form the basis of the proposed architecture. Behavior Management based architecture has been identified as the strategic basis. This led to the introduction of a new layer called the behavior management layer in the agent architecture in addition to the interface and behavioral layers. Since the architectures for interface and behavior layers have already been established in the agent literature, the architecture of the behavior management layer and how it fits with the interface and the behavioral layers are described.

Since behavior management calls for a self-managing behavior of an agent, the architecture has been described by considering the properties that have to be fulfilled by a self-managing system. As is the case with the proposed paradigm described in the previous chapter, the behavior management architecture is also a generic one and is required to elevate any function performed by the agent into a functional faculty of an agent.

Finally, the architecture has been evaluated using the SAAM methodology to validate the appropriateness of the architecture.
The main contributions of this chapter are as follows:

- Proposing a Behavior Management Architecture for the language faculty of agents.
- Providing solution for the issues identified in the proposed abstract self-management architecture.
- Providing an architecture which can fulfill not only the current requirements but also the foreseen requirements with simple to medium efforts as indicated in the evaluation.