important when borrowing a method from a library of reusable components of problem solvers. It is important to note that the fact that any concepts are used within the framework of the method does not mean that the formal record of their definitions is part of this method. For example, a method that allows solving problems for calculating the area of a triangle will include various formulas for calculating the area of a triangle but will not include the definitions of the concepts ‘area”, ‘triangle”, etc., since if there are a priori correct formulas, these definitions will not be used directly in the process of solving the problem. At the same time, the formal definitions of these concepts will be part of the declarative semantics of this method.

Combining the *method* and its operational semantics, that is, information about how this *method* should be interpreted, we will call a ***skill***.

skill
- := [an ability]
- := [a combination of a *method* with its comprehensive specification – a *complex representation of the operational semantics of the method*]
- := [a method + a method of its interpretation]
- := [an ability to solve the corresponding class of equivalent problems]
- := [a method plus its operational semantics, which describes how this method is interpreted (performed, implemented) and is at the same time the operational semantics of the corresponding problem solving model]
- ⇒ *subdividing**:
  {
  - *active skill*
    - := [a self-initiating skill]
  - *passive skill*

  }

Thus, the concept of a *skill*is the most important concept from the point of view of constructing problem solvers, since it combines not only the declarative part of the description of the method of solving a class of problems but also the operational one.

Skills can be *passive skills*, that is, such *skills*, the usage of which must be explicitly initiated by some agent, or *active skills*, which are initiated independently when a corresponding situation occurs in the knowledge base. To do this, in addition to the *method*and its operational semantics, the *sc-agent*, which responds to the appearance of a corresponding situation in the knowledge base and initiates the interpretation of the *method*of this *skill*, is also included in the *active skill.*

This separation allows implementing and combining different approaches for solving problems, in particular, *passive skills* can be considered as a way to implement the concept of a smart software package.

VIII. Concepts of a class of methods and a language for representing methods

Like actions and problems, methods can be classified into different classes. We will define a set of methods, for which it is possible to unify the representation (specification) of these methods, as a ***class of methods***.

class of methods
- ⇐ *family of subclasses**:
  method
- := [a set of methods, for which the representation language of these methods is set]
  - ∋ *procedural method for solving problems*
    - ⊃ *algorithmic method for solving problems*
  - ∋ *logical method for solving problems*
    - ⊃ *productional method for solving problems*
    - ⊃ *functional method for solving problems*
  - ∋ *artificial neutral network*
    - := [a class of methods for solving problems based on artificial neural networks]
  - ∋ *genetic ”algorithm”*
- := [a set of methods based on a common ontology]
- := [a set of methods represented in the same language]
- := [a set of methods for solving problems, which corresponds to a special language (for example, an sc-language) that provides a representation of methods from this set]
- := [a set of methods that corresponds to a separate problem-solving model]

Each specific *class of methods* mutually identically corresponds to a *language for representing methods* that belong to this (specified) *class of methods.* Thus, the specification of each *class of methods* is reduced to the specification of the corresponding *language for representing methods*, i.e., to the description of its syntactic, denotational and operational semantics.

Examples of *languages for representing methods* are all *programming languages*, which mainly belong to the subclass of *languages for representing methods* – to *languages for representing methods for information processing.* But now the need to create effective formal languages for representing methods for performing actions in the environment of cybernetical systems is becoming increasingly relevant. Complex automation, in particular, in the industrial sphere, is impossible without this.

There can be a whole set of such specialized languages, each of which will correspond to its own model of problem solving (i.e., to its own interpreter).

language for representing methods
- := [a method language]