

# Agent-based Modelling and Simulation in Simio

## Abstract - Master thesis

IT Faculty

University of Applied Sciences Rapperswil

Fall term 2017 and spring term 2018

Author:	Thomas Kehl
Examiner:	Prof. Dr. Andreas Rinkel
Expert:	Knut Schmahl
Deadline:	19 August 2018

## Abstract

### Initial situation

Agent-based modelling and simulation is a relatively modern approach to the modelling of systems. It enables the modelling of the dynamics of complex and cybernetic systems. They are often self-organising systems, which create emerging effects, such as for example the escape behaviour of human beings.

The software Simio enables the discrete simulation of events. However, the software does not offer options for agent-based simulations. The strength of Simio consists in the object-oriented implementation of discrete event simulation. On this basis, the requirements for an agent-based simulation in Simio were developed in the preparatory phase of this thesis. In this context, the following main aspect became apparent:

Basically, the aim is to develop an agent-object. This object comprises the following central features:

- 1) It should be able to contain its own logistics.
- 2) It should be able to recognise the surrounding.
- 3) Agent-objects should be able to communicate with each other.

The entity-object serves as the basis for this agent-object. Using the approach of the steering behavior Simio enables the development of complex behaviour logic for entities, as soon as these are in the free space. This combination serves as a basis for the implementation of agent-based modelling and simulation for Simio as part of the master thesis.

For the configuration or the development of the agents' logic during the modelling (design time) there is the option to create AddIns in Simio. Subsequently, an AddIn is developed which provides the graphic interface for the modelling and configuration of the agent (see Figure 1).

### Implementation

Two new central objects are developed in Simio:

- AgentBase
- ComplexGateway

Additionally, the AddIn "SimioAgentLibrary" was developed to configure the agent.

The steering behaviour "Agent" is however the main component responsible for the implementation of the agent logic.

Below, please find a brief overview for every element.

#### Object: AgentBase

As already mentioned the object "AgentBase" is based on the Simio object "Entity". And it has the centre feature "SteeringBehavior". It is used to assign the SteeringBehavior "Agent" to the object. Additionally, it is possible to define a standard Update TimeInterval for the implementation of the simulation steps for the Steering Behavior.

#### Object: ComplexGateway

The object "ComplexGateway" is based on the Simio object "Fixed". It mainly serves to transfer an agent (AgentBase) into the FreeSpace. With this transfer the SteeringBehavior is activated. For the purpose the object has an InputNode and an OutputNode. Additionally, the capacity can be configured to determine how many agents can be in the ComplexGateway at the same time. In relation to this functionality the nodes have a buffer each, so that the agents can be buffered when ComplexGateway is busy.

**Combination DES and ABS:** The object “ComplexGateway” enables to mix and combine the approach of the discrete event simulation (DES) with the one of the agent-based simulation (ABS). Specifically, an agent can be part of a discrete, process-oriented event simulation just as any other entity in Simio. When an agent reaches a ComplexGateway during the simulation, it is transferred into the FreeSpace. The agent will then act as an agent, and its own logic is used and processed which has been implemented using the AddIn “SimioAgentLibrary”.

### AddIn: SimioAgentLibrary

The AddIn “SimioAgentLibrary” provides the interface of the development of the agent logic (see Fig. 1 – in the following paragraphs there are detailed descriptions and illustrations of the components). The logic is mainly developed as a StateMachine. The different actions of the states and conditions are implemented with C#. These implementations can be completed directly on the interface. Alternatively, there is the option to include and use additional NET assemblies. This means, the entire “.NET world” is available for the implementation of the agent logic.

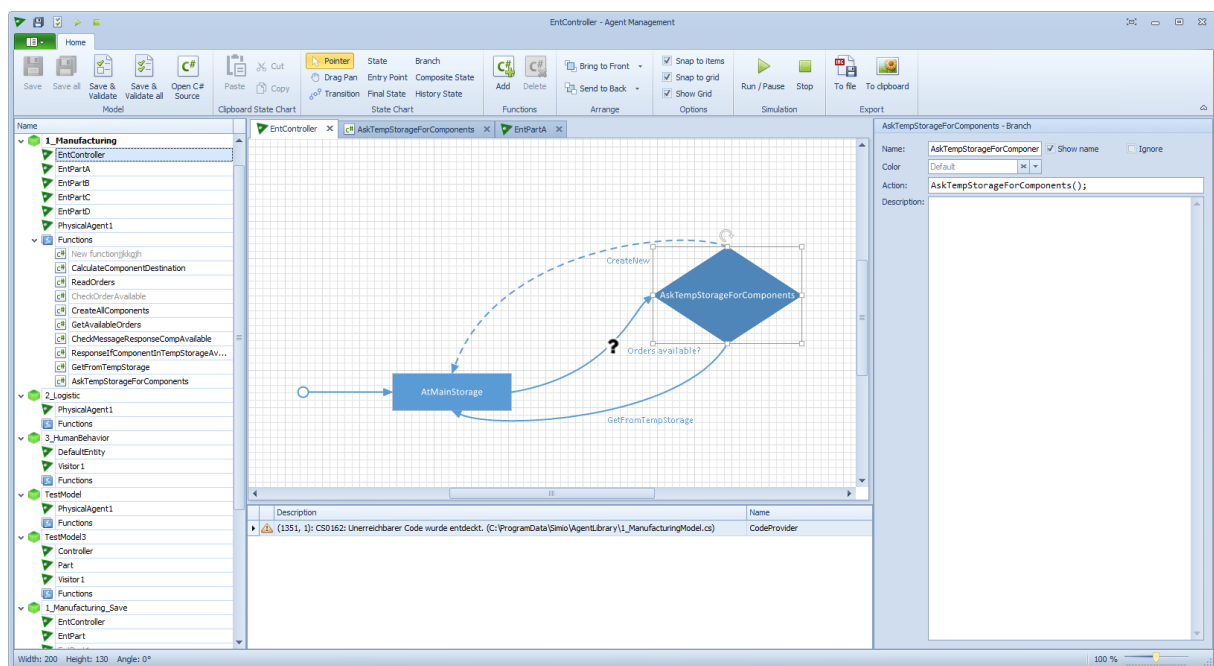


Figure 1: AddIn SimioAgentLibrary

The interface consists of the following components:

#### 1) Model tree

This tree lists all models available in the project currently opened in Simio (see Figure 2). The agents (i. e. all objects of the basic type “AgentBase”) are listed for every model. Additionally, there is a “Functions” area for every model. Here, complete functions (more exact methods) can be implemented using C#. These functions are afterwards available through the use in the actions of status and transitions. Functions in grey in Figure 2 are set to inactive.

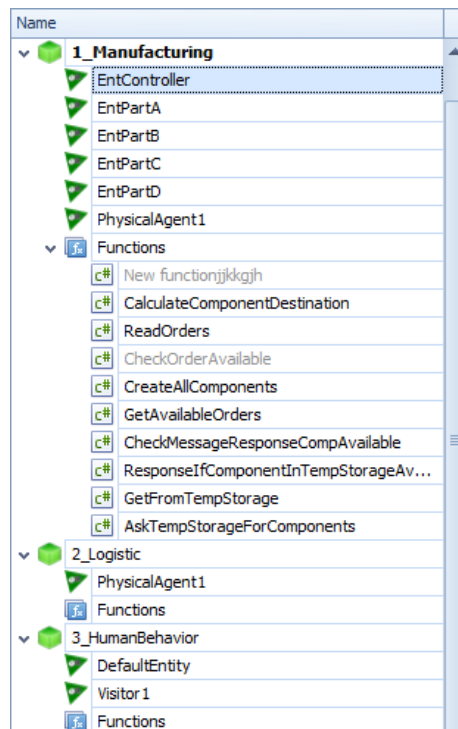


Figure 2: Model tree

## 2) Processing area

In this area the StateMachine can be modelled graphically (see Figure 3). Functions can also be programmed in the processing area. This is shown in the left part of Figure 6.

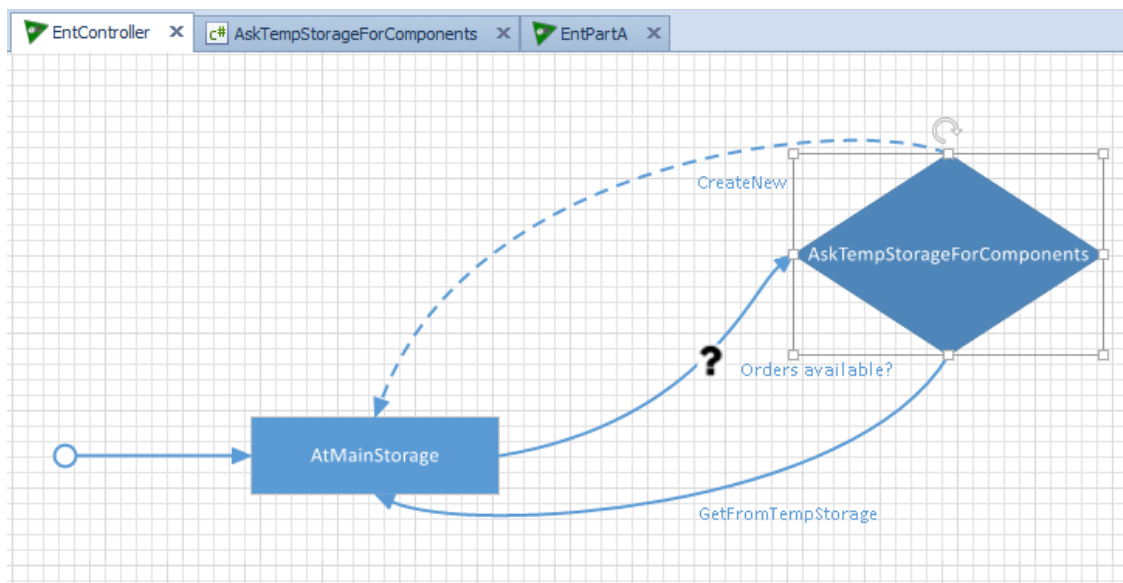


Figure 3: Processing area with StateChart

For the modelling of the StateMachine the main menu provides the tools as shown in Figure 4. The user can insert them in the processing area by means of Drag&Drop.

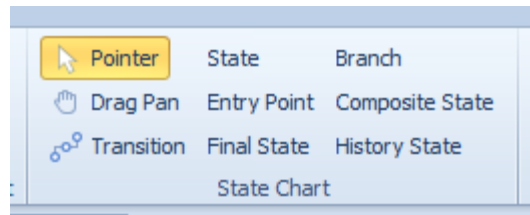


Figure 4: Tools for StateChart

### 3) Configuration area

Here the element selected in the processing area can be configured. It is for example possible to implement the actions for Entry and Exit for a state (see Figure 5). As already mentioned before, the implementation is done with C#. If one line is sufficient, the logic can be inserted directly into the corresponding action. Otherwise, there is the option to create and insert a function. It is, however, also possible to use any code from any referenced assembly.

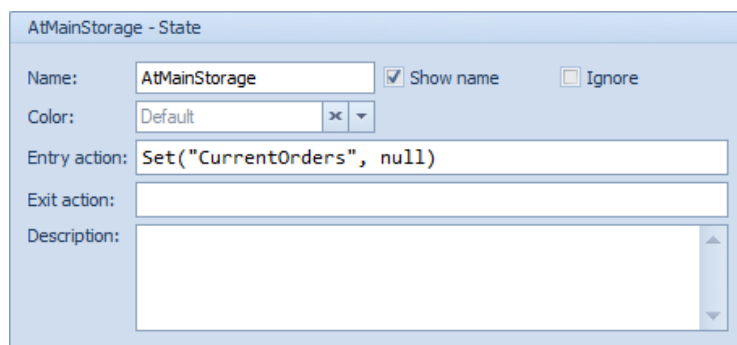


Figure 5: Configuration of a state

If a function is active in the processing area, there is the option to configure the desired arguments as well as the type of return of the function (see Figure 6).

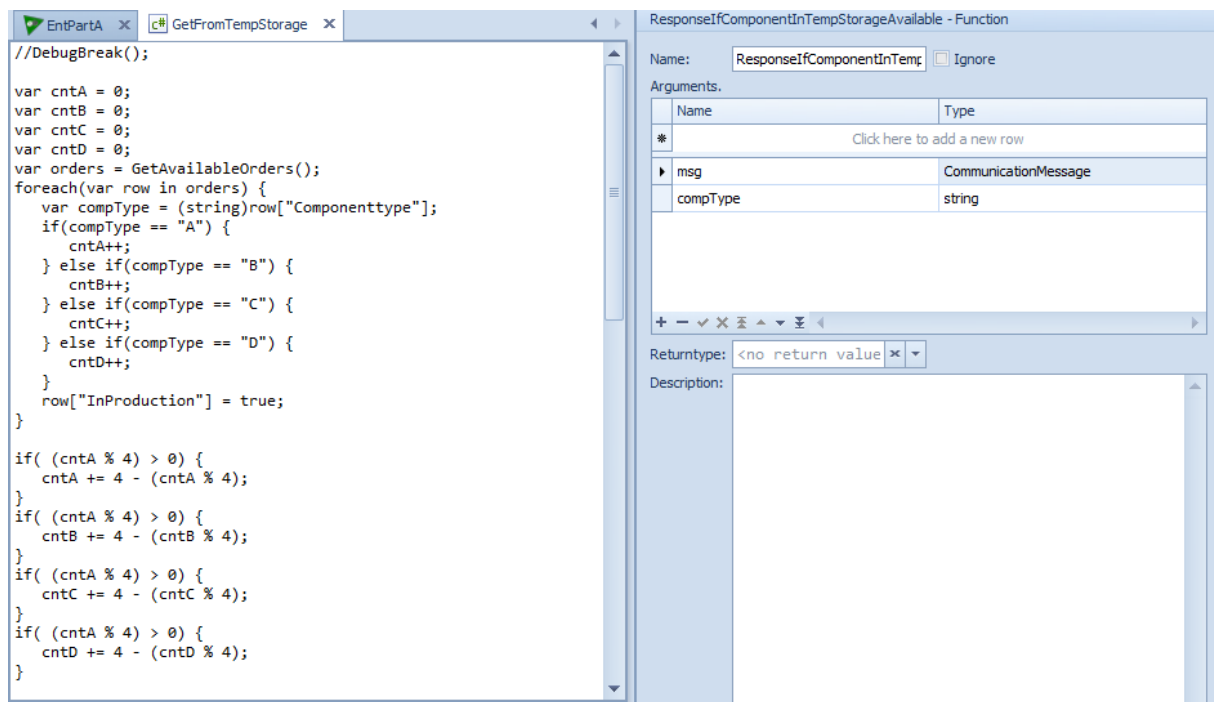


Figure 6: Implementation and configuration of a function

#### 4) Main menu

The main menu offers various options:

- Saving the StateMachine – these are saved in Simio in the current project. This means, no additional files are created for their persistence.
- Validation - this enables the valuation of the C# codes for the current model or for all models with regard to their syntactic validity. Errors are displayed in the notification area (see Figure 8)
- Open C# Source – this menu item offers the option to display the entire C# code that is compiled for the execution of the simulation. The line references for errors in the notification area refer to this code file. This enables a complete overview and facilitates the error search.
- Elements for StateChart – see “Processing area” or Figure 4
- Functions – adding or deleting functions
- Simulation – start/stopping/pausing the simulation of the current model.

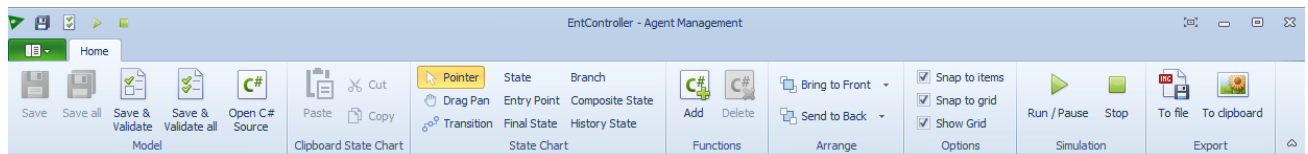


Figure 7: Main menu

#### 5) Notifications

This list contains notifications that have been generated in different places.

	Description	Name
▶	(251, 13): CS0201: Nur assignment-, call-, increment-, decrement-, await- und 'new object'-Ausdrücke können als Anweisung verwendet werden. (C:\ProgramData\Simio\AgentLibrary\1_Manufacturing_SaveModel.cs)	CodeProvider
✖	(251, 13): CS0122: Der Zugriff auf 'a' ist aufgrund der Sicherheitsebene nicht möglich. (C:\ProgramData\Simio\AgentLibrary\1_Manufacturing_SaveModel.cs)	CodeProvider
✖	(251, 13): CS0118: 'a' ist ein(e) 'Typ', wird aber wie ein(e) 'Variable' verwendet. (C:\ProgramData\Simio\AgentLibrary\1_Manufacturing_SaveModel.cs)	CodeProvider
✖	(258, 13): CS0201: Nur assignment-, call-, increment-, decrement-, await- und 'new object'-Ausdrücke können als Anweisung verwendet werden. (C:\ProgramData\Simio\AgentLibrary\1_Manufacturing_SaveModel.cs)	CodeProvider
✖	(258, 13): CS0122: Der Zugriff auf 'b' ist aufgrund der Sicherheitsebene nicht möglich. (C:\ProgramData\Simio\AgentLibrary\1_Manufacturing_SaveModel.cs)	CodeProvider
✖	(258, 13): CS0118: 'b' ist ein(e) 'Typ', wird aber wie ein(e) 'Variable' verwendet. (C:\ProgramData\Simio\AgentLibrary\1_Manufacturing_SaveModel.cs)	CodeProvider
✖	(265, 13): CS0201: Nur assignment-, call-, increment-, decrement-, await- und 'new object'-Ausdrücke können als Anweisung verwendet werden. (C:\ProgramData\Simio\AgentLibrary\1_Manufacturing_SaveModel.cs)	CodeProvider
✖	(265, 13): CS0122: Der Zugriff auf 'c' ist aufgrund der Sicherheitsebene nicht möglich. (C:\ProgramData\Simio\AgentLibrary\1_Manufacturing_SaveModel.cs)	CodeProvider

Figure 8: Notification display

#### SteeringBehavior: Agent

The implementation of the SteeringBehavior “Agent” is the core of the implementation. If there is an agent in a ComplexGateway, it has been transferred into FreeSpace. Subsequently, the behaviour of the agent is controlled by its SteeringBehavior. The SteeringBehavior “Agent” comprises the implementation of a generic StateMachine that loads the configuration (i. e. the specific modelled StateMachine, see also Figure 10 – incl. all features etc.) of the agent and controls the agent according to this model. The model is loaded during the first input into a ComplexGateway and executed according to the EntryPoint. When the agent leaves the ComplexGateway, it is returned into a discrete event simulation. The agent’s state is, however, maintained and is continued when it enters the next ComplexGateway.

An API is available to control the agent. Below, please find an extract of the API functionalities.

```
public interface IAPI {
    API.IAgent Agent { get; }
    TimeSpan ExecutedTime { get; }
    DateTime CurrentTime { get; }
    void FireEvent(string eventname);
    void Set(string key, object value);
    T Get<T>(string key);
    T Get<T>(string key, T def);
    void BasePath { get; }
}

public interface IAgent {
    string Name { get; }
    void MoveTo(Vector2 target);
    void MoveTo(Vector3 target);
    void MoveToNode(string nodename);
    void JumpTo(Vector2 target);
    void Send(object msg, AgentBase dest, string replyWith = null);
    bool TryGetNextMessage(out CommunicationMessage message);
    void SetTimeInterval(int interval);
    void ResetTimeInterval();
}
```

Table 1 Contains the explanations of several selected functions.

Function	Description
<code>IAPI.FireEvent()</code>	Triggers a certain event in Simio in the current model. This also enables the agent to carry out the processes in Simio.
<code>IAPI.BasePath</code>	Returns the basic path of the Simio project file. This is helpful to feed in data in relation to the project path with C#.
<code>IAgent.MoveTo()</code>	Triggers that the agent is “running” to a certain point in the coordination system.
<code>IAgent.MoveToNode()</code>	Triggers that the agent leaves FreeSpace and moves on to a certain node in the network.
<code>IAgent.Send()</code>	Sends a message to one or several agents.
<code>IAgent.SetTimeInterval()</code>	Changes the time interval of the simulation for the current agent.

**Table 1: Functionality of API (extract)**

Obviously, API can be expanded, if necessary.

With the result, it is possible to create agent-based models in Simio as well to carry out their simulation. As already mentioned, the agent-based simulation can also be combined with the process-oriented discrete event simulation known in Simio.

Figure 9 shows a Simio model. The object *Decision* corresponds to a ComplexGateway. Figure 10 shows the StateChart which is activated when an agent reaches the object *Decision*.



The `EntryPoint` refers to a state of the branch type. Depending on the conditions the agent advances either to the sink *SnkMainStorage*, the sink *SnkLogistics* or the ComplexGateway *TempStorage*. If the condition for *ToTempStorage* applies, the API function `IAgent.MoveToNode("Input@TempStorage")` is executed. This triggers that the agent leaves the ComplexGateway and advances to the InputNode of the ComplexGateway *TempStorage*. Once the agent has arrived, it waits until it receives the message `AskForComponentInTempStorage` or `ToProduction` (These messages are sent by the agent *EntController* that is in the ComplexGateway *MainStorage* during the simulation – and thus carries out its agent logic, which is also shown in Figure 3). The agent responds to the message `AskForComponentInTempStorage` correspondingly and returns to the state *AtTempStorage*. As a response to the message `ToProduction` the agent carries out the function `IAgent.MoveToNode("TNodeStart")` that triggers that the agent advances to the corresponding node.



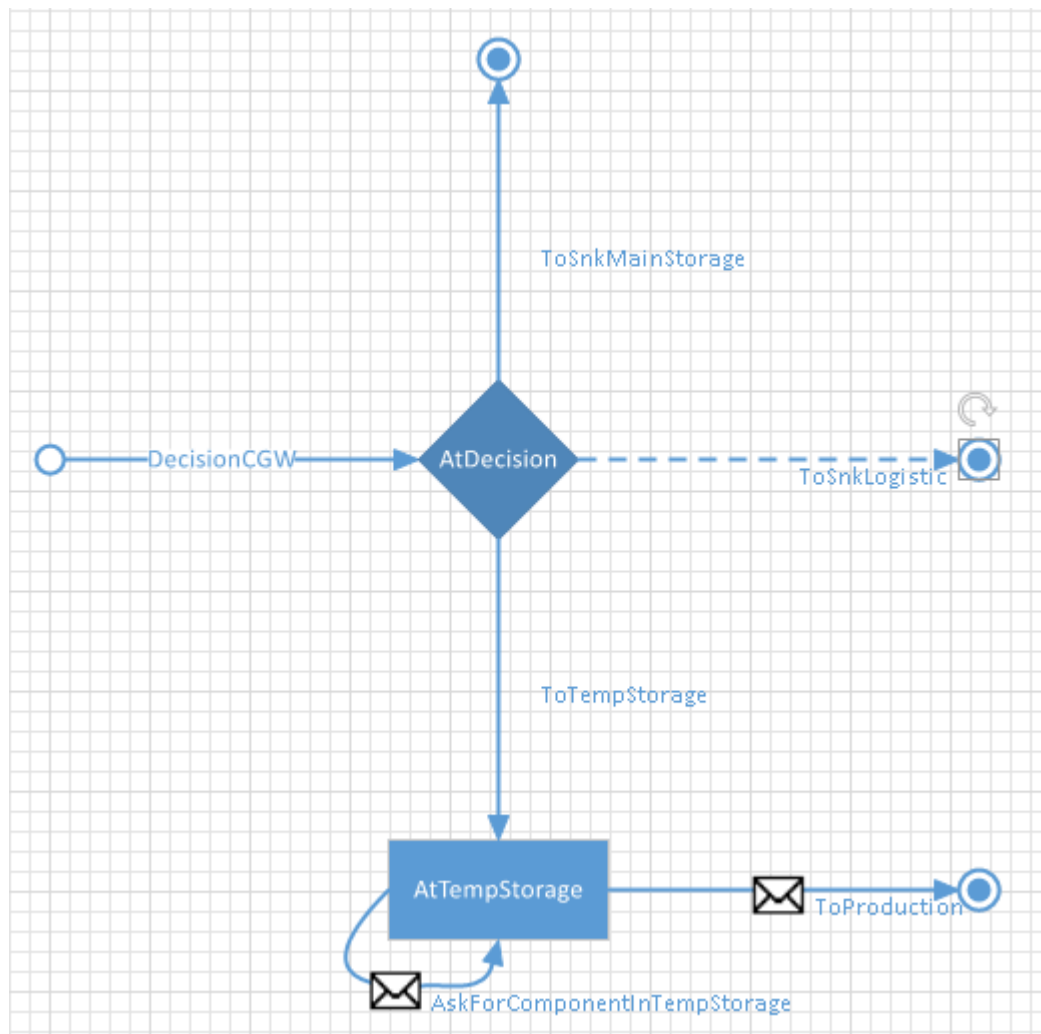


Figure 10: StateMachine model of an agent

## Outlook

This implementation enables comprehensive agent-based modelling and simulation in Simio by a user that is both apt in the use of Simio and also has a basic knowledge of C# to be able to create a complex logic, if needed. For any other steps there are two further approaches:

- 1) Depending on the requirements, the API is expanded.
- 2) The UI for the AddIn "SimioAgentLibrary" is expanded further for example to support IntelliSense etc.