KidSim: Agenten Sprachlos Programmieren

1. Einführung:

Wenn man sich mit dem Thema "Agenten sprachlos programmieren" beschäftigt (wobei sprachlos hier bedeutet: ohne den Einsatz traditioneller Computersprachen), stößt man auf zwei Fragen. Erstens, warum sollte ein Anwender[1] überhaupt selber einen Agenten programmieren. Und zweitens, da Anwender in der Regel nicht programmieren können, bleibt die Frage, wie ein solcher Agent von einem Anwender ohne den Einsatz von Programmiersprachen programmiert werden kann.

2. Agenten als persönliche Assistenten

In der heutigen Computer−Welt gibt es drei wichtige Entwicklungen:
−immer mehr alltägliche Aufgaben werden mit dem Computer erledigt,
−die zur Verfügung stehende Information ist dynamisch und unstrukturiert und nimmt explosionsartig zu (verfügbar über das Internet),
−die Anzahl der Nutzer von elektronisch verfügbaren Informationen mit begrenzten Computerkenntnissen nimmt ebenfalls stark zu.
Unter diesen Bedingungen können Agenten eine wichtige Rolle als persönliche Assistenten der Menschen spielen. Sie sind intelligente User−Interfaces, die den Anwendern helfen, indem sie mit anderen Applikationen interagieren und autonom Aufgaben ausführen. Der Anwender kann Aufgaben an den Agenten delegieren, der die Interessen, Gewohnheiten und Vorlieben seines Besitzers kennt. Der Agent kann Vorschläge machen oder an Stelle des Anwenders agieren, ohne dass dieser aktiv werden muss. Um all diese Funktionen zu erfüllen, benötigen diese persönlichen Assistenten sehr gute Anpassungsfähigkeiten an den jeweiligen Anwender, um ihm genau die Hilfe anzubieten, die er gerade braucht (Maes, 1994).
Die Hilfe eines persönliche Assistenten bringt viele Vorteile für den Anwender. Er bekommt Unterstützung bei langweiligen oder zeitfressenden Arbeiten, genau dann, wenn er diese haben will. Er kann sich auf die wesentlichen Arbeiten konzentrieren, der Agent erledigt den "Rest". Außerdem kann er mit einer größeren Menge von Information umgehen (Wagner, 2000). Beispiele für Agenten als persönliche Assistenten sind beispielsweise Open Sesame vom MIT Media Lab oder Eager von Stagecast (Wagner, 2000). Open Sesame ist ein persönlicher Assistent, der z.B. für das Dateimanagement und als e−mail Filter einsetzbar ist. Open Sesame basiert auf maschinellen Lerntechniken. Eager ist ein intelligenter persönlicher Agent. Er beobachtet den Anwender, wenn dieser sich wiederholende Aktionen ausführt, schreibt Eager ein Programm, der diese Aktivität ausführt (Programming by Demonstration, "smart macro recorder").
Da diese persönlichen Assistenten größtenteils die gleichen Aufgaben haben, sind sie sich sehr ähnlich. Sie werden daher als Software−Entitäten entweder kopiert oder mit Hilfe von Agent−Frameworks spezifiziert. Dadurch kann sich jeder Anwender selbst seinen persönlichen, personalisierten Assistenten herstellen (Wagner, 2000).

[1] Anwender (= End User): Menschen die Computer benutzen, aber keine professionellen Programmierer sind und in der Regel nicht programmieren, sondern Applikationen benutzen (Smith et al., 1994).

Die Frage bleibt, wie sich die Agenten an den Anwender anpassen können. Auf welche Weise kann ein Agent lernen, wie er sich in einer gegebenen Situation verhalten soll. Dafür gibt es verschiedene Möglichkeiten (Maes, 1994):

− Maschinelles Lernen

Der Agent hat ein minimales Hintergrundwissen und lernt ein angemessenes Verhalten durch die Beobachtung des Anwenders (oder anderer Agenten). Er erkennt Muster im Verhalten des Anwenders und wertet diese aus (beobachten und imitieren) oder lernt durch Interaktion mit dem Anwender (oder mit anderen Agenten, die ähnliche Aufgaben ausführen).

> Vorteile: Der Agent verbessert seine Fähigkeiten ständig, ohne eine große, initiale Wissensbasis zu benötigen. Der Agent passt sich sehr genau dem einzelnen Anwender an.

Nachteile: Es dauert lange, bis ein Agent sich der Umgebung und dem Anwender angepasst hat und tatsächlich von Nutzen ist. Der Agent kann nur existierende Muster wiederholen.

− Wissensbasierter Ansatz (traditionelle AI−Techniken)

Der Agent hat fachspezifisches Hintergrundwissen, er ist mit einer umfassenden Wissensbasis über Anwender und Umgebung ausgestattet.

> Vorteile: Es gibt ein homogenes Interface für alle Anwender. Ein Agent ist von Beginn an, ohne Einsatz von Zeit und Aktivität von Seiten des Anwenders, einsatzfähig.

Nachteile: Der Agent benötigt eine sehr große Wissensdatenbank, er ist entsprechend aufwendig zu programmieren. Der Agent kann sich nicht an den einzelnen Anwender anpassen.

− Anwender−Programmiert

Der Anwender stellt selber die Regeln und Kriterien auf, nach denen der Agent agiert.

Vorteile: Er kann dem Agenten vertrauen, dass dieser sich in einer gegebenen Situation auf eine bestimmte Weise verhält. Der Agent ist sofort adaptiert an die individuellen Bedürfnisse des Anwenders.

> Nachteile: Die Programmierung und Betreuung eines Agenten ist für viele Anwender zu schwierig. Sie erkennen nicht immer, wann sie einen Agenten einsetzen können. Dieser Ansatz bewährt sich unter statischen Umgebungsbedingungen, nicht aber unter dynamischen.

Der Mechanismus des maschinellen Lernens ist allgemein anerkannt, um autonome Agenten zu entwickeln. Um den Einsatz des Agenten von Anfang an zu verbessern, wird er dabei im Idealfall mit den beiden anderen Mechanismen kombiniert (d.h. der Agent hat Zugang zu Hintergrundwissen, er kann vom Anwender lernen, der Anwender kann den Agenten aber auch explizit programmieren, z.B. wenn der Agent neu ist oder der Anwender das Verhalten des Agenten ändern möchte).

Ein wichtiger Punkt bei der Erschaffung eines persönlichen Agenten ist also die Programmierung durch den Anwender. Nach Smith et al. (1994) ist dies sogar der beste Weg, um eine maßgeschneiderte Software−Entität zu erhalten. Das Problem ist allerdings, dass nur ein geringer Teil der Anwender die nötigen Fähigkeiten hat, um selber Agenten mit traditionellen Programmiersprachen zu entwickeln.

3. Das End User Programming Problem

Die meisten Menschen heutzutage können mit PC’s umgehen. Millionen Menschen benutzen täglich den Computer um Texte zu schreiben, e−Mails zu verschicken, im Internet zu surfen, Computerspiele zu spielen u.a.m.. Computer sind nicht an sich unbenutzbar. Trotzdem können nur die wenigsten Anwender programmieren (End User Programming Problem), die meisten können eben nur "anwenden". Das heißt aber auch, dass nur die wenigsten Menschen in der Lage wären, einen Agenten zu programmieren. Es kann verschiedene Gründe geben, warum nur wenige Menschen programmieren können. Zum einen, weil das Programmieren

an sich zu schwierig ist oder zum anderen, weil die existierenden Programmiersprachen zu kompliziert sind. Die meisten Menschen können einem vorgegebenen Rezept folgen, Direktiven geben, sich Abläufe und Situationen vorstellen, Reisen planen, also mentale Aktivitäten ausführen, die auch beim Programmieren benötigt werden. Auch die grundlegenden Konzepte der Programmierung (sequentielle Abarbeitung, bedingte Anweisungen, Schleifen) werden von jedem verstanden (z.B. beim Kochen nach Rezept). Das heißt, dass nicht das Programmieren an sich zu kompliziert ist, sondern dass die Programmiersprachen zu schwer sind (Smith et al., 1994).
Es gab in der Vergangenheit verschiedene Lösungsansätze um das End User Programming Problem zu lösen (Cypher, 1993):
1. Einstellungen (Preferences)
Viele Applikationen sind über Einstellungen konfigurierbar. Sie bieten verschiedene Möglichkeiten an, wie sie an die Bedürfnisse eines Nutzers angepasst werden können. Allerdings sind diese Einstellungen an Situationen gebunden, die sich ein Software−Designer vorstellen kann. Eine völlige Anpassung einer Applikation an die Bedürfnisse eines Anwenders ist so nicht möglich. Es kann in der Regel nur ein kleiner Teil der einstellbaren Möglichkeiten berücksichtigt werden.
2. Script Sprachen
   Script Sprachen sind sehr einfache Programmiersprachen. Trotzdem bleibt das Hauptproblem, dass der Anwender immer noch die Grundlagen von Programmiersprachen und die Konventionen der Sprache erlernen muss.
3. Macro Recorder
   Der Anwender führt Aktionen durch und der Macro Recorder zeichnet diese auf. Die aufgezeichnete Befehlssequenz kann später wiederholt werden. Macro Recorder sind nur für eng begrenzte Aufgaben einsetzbar.

Diese drei Ansätze konnten das End User Programming Problem nicht lösen. Eine Beobachtung ist, dass Anwender häufig Editoren benutzen. Hier erschafft der Anwender Objekte durch eine Sequenz von Aktionen, kann diese Objekte verändern und stoppt, wenn ihm das Resultat zusagt. Eine Idee war daher, das Programmieren so einfach zu machen wie das Editieren (dasselbe Prinzip wie beim Macro Recorder). Entscheidend ist also, mit einer graphischen Benutzeroberfläche zu programmieren.

4. Programmieren mit graphischer Oberfläche

Um mit einer graphischen Benutzeroberfläche zu programmieren, war es wichtig, die Prinzipien von GUI's (Graphical User Interfaces) auf das Programmieren zu übertragen. Die dafür wichtigsten Prinzipien sind (Smith et al., 1994):
−Sichtbarkeit: Alles für den Anwender Relevante sollte auf dem Bildschirm (als Objekt) sichtbar sein.
−Sehen und zeigen: Man sollte Objekte auf dem Bildschirm sehen und diese direkt anwählen und manipulieren können (z. B. mit dem Mauszeiger).
−Kopieren und modifizieren: Objekte sollten kopier− und modifizierbar sein.
−Konkret statt abstrakt: Software−Entitäten sollten konkret (als Objekt) repräsentiert sein.
−Minimale Übersetzungsdistanz: Minimierung der Distanz zwischen der menschlichen mentalen Repräsentation von Konzepten und der Repräsentation im Computer.

Die GUI−Prinzipien eliminieren Kommandozeilen und ersetzen sie durch die visuelle Repräsentation von Objekten und der direkten Manipulation dieser Repräsentationen. Alle erfolgreichen Editoren folgen diesem Ansatz, die meisten Programmierumgebungen tun es nicht.
Nach Smith et al. (1994) müssen außerdem zwei Konzepte umgesetzt werden, um mit Hilfe einer graphischen Benutzeroberfläche zu programmieren:

1) Graphical Rewrite Rules
Zweidimensionale Varianten von Rewrite Rules[2]. Eine Graphical Rewrite Rule beschreibt eine Transformation eines diskreten Teils des Bildschirms von einem Zustand in einen anderen. Jede Graphical Rewrite Rule besteht aus zwei Teilen: einem "vorher"− und einem "nachher"−Teil. Eine Regel wird angewandt, wenn der "vorher"−Teil der Regel einem Teil des Bildschirms zur aktuellen Zeit gleicht. Wenn eine Regel angewendet wird, wird diese Region in den "nachher" −Zustand umgewandelt.
Graphical Rewrite Rules funktionieren gut für einfache Aufgaben, haben aber drei Nachteile:
−Das Rule−Generality Problem: es ist schwierig, Bilder verallgemeinert auf verschiedene Situationen anzuwenden.
−Das Rule−Semantics Problem: es ist schwierig graphisch festzulegen, wie der Computer die Transformation von der linken zur rechten Seite der Regel durchführen soll.
−Das Rule−Sequencing Problem: es ist schwierig, eine Serie von Transformationen (wenn aus A B folgt, folgt aus B C) zu beschreiben, weil die Regeln unabhängig voneinander sind.
2) Programming By Demonstration
Das System wird in "Aufnahme" − Modus gesetzt, der Anwender führt Aktionen durch, das System nimmt die Aktionen des Anwenders auf und wandelt sie in ein durchführbares Programm um (wie beim Macro Recorder). Wichtig ist, dass die Anwender mit dem System interagieren, als ob das "aufnehmen" nicht stattfände, d.h. es muss nichts neues gelernt werden, um diese Technik einzusetzen.
Beim Programming By Demonstration tauchen zwei Probleme auf. Erstens ist die Frage, inwieweit die demonstrierten Aktionen des Anwenders abstrahiert werden können. Daher ist diese Technik bisher nur für die Programmierung relativ einfacher Probleme benutzt worden. Zweitens werden die aufgezeichneten Programme nicht in einer für den Anwender verständlichen Form, sondern in der Regel als Scripte repräsentiert (Representation Problem).

Eine Programmierumgebung, in der die GUI−Prinzipien und diese beiden Konzepte (Graphical Rewrite Rules, Programming By Demonstration) konsequent umgesetzt werden, ist KidSim. Dieses Programm benutzt keine traditionelle Syntax, Regeln werden graphisch, nicht abstrakt repräsentiert.

5. Programmbeispiel: KidSim

KidSim (Kid's Simulation) wurde von Allen Cypher Anfang der neunziger Jahre für die Firma Apple entwickelt. Inzwischen hat Allen Cypher eine eigene Firma (Stagecast) gegründet und bringt KidSim unter dem Namen Stagecast Creator auf den Markt.
KidSim ist eine Programmierumgebung für Kinder um Mikro−Welt Simulationen zu implementieren. Kinder können ihre eigenen symbolischen Simulationen[3] erschaffen, verändern und laufen lassen. Sie können ihre eigenen Charaktere (Agenten) erschaffen und sie erschaffen Regeln die festlegen, wie sich diese Charaktere verhalten und wie sie interagieren (damit ist KidSim ein Agent Framework). KidSim gehört zu einer neuen Generation von komplett graphischen Programmierumgebungen für Kinder, in denen Kinder durch die direkte Manipulation von bildlichen Objekten programmieren. Um damit zu programmieren muss man keine konventionelle Programmier− oder Scriptsprache mehr erlernen. Das Ziel von KidSim ist es, dass Kinder über das Spielerische implementieren und modifizieren von simulierten Welten die grundlegenden Fähigkeiten erlernen, die nötig sind, um programmieren zu können (Cypher & Smith, 1995).

[2] Rewrite Rule: if−then−Regel, Grundlage von Produktionssystemen (regelbasierten Systemen, z. B. Expertensystemen). Die Regeln sind voneinander unabhängig, man kann neue Regeln hinzufügen, ohne die alten Regeln zu beeinflussen (dabei darf man keine Regeln überschreiben) und das System muss die Regeln in der korrekten Reihenfolge ausführen (Smith et al., 1994).

[3] Eine symbolische Simulation ist hier eine Computer−kontrollierte Mikro−Welt, die aus individuellen Objekten (Agenten) besteht, die sich auf dem "Spielbrett" umherbewegen und mit anderen Objekten interagieren können (Smith et al., 1994).

Mit KidSim kann man Mikrowelt−Simulationen, einfache Spiele (z.B. PacMan), aber auch Implementierungen von Algorithmen (z.B. den Siebalgorithmus des Eratosthenes, Simulation einer Turing−Maschine) programmieren.
Die wichtigsten Bestandteile einer KidSim Simulation sind (Smith et al., 1994):
− das Spielbrett, das in diskrete Regionen aufgeteilt ist (repräsentiert die simulierte Mikrowelt);
− der Rundenzähler (Uhr), startet und stoppt Simulationen, unterteilt die Zeit in diskrete Runden, die Simulation läuft sequentiell in Runden, jede Runde wird für jedes Objekt überprüft, welche Regeln anzuwenden sind;
− die Copy Box, Quelle neuer Simulations−Objekte ;
− der Regel−Editor, die Regeln werden durch Programming By Demonstration definiert und durch Graphical Rewrite Rules gespeichert;
− die simulierten Objekte (Agenten):
"Ein (KidSim−) Agent ist eine persistente Software−Entität, die einem speziellen Zweck dient. ‚Persistent' unterscheidet Agenten von Subroutinen; Agenten haben ihre eigenen Ideen, wie eine Aufgabe auszuführen ist, ihre eigenen Ziele. Der ‚spezielle Zweck' unterscheidet sie von Multifunktions−Applikationen, Agenten sind typischerweise viel kleiner." (Smith et al., 1994)
Alle aktiven Objekte in der Simulation sind Agenten (Charaktere). Sie können sich auf dem Spielfeld bewegen und mit anderen Agenten interagieren. KidSim−Agenten besitzen (Cypher & Smith, 1995):
−Aussehen: graphische Repräsentation auf dem Bildschirm, ein Objekt kann verschiedene Aussehen haben, die sich nach definierten Regeln ändern.
−Eigenschaften (Properties): repräsentieren Eigenschaften des Charakters, die nicht sichtbar sind (entsprechen Instanzvariablen), z.B. Hunger, Alter, Fähigkeiten. Durch Regeln können die Werte der Eigenschaften verändert werden.
−Regeln: diese definieren das Verhalten des Agenten in einer gegebenen Situation.

Bei KidSim werden die speziellen Probleme der implementierten Konzepte (Graphical Rewrite Rules, Programming By Demonstration) folgendermaßen gelöst:
−Rule−Generality Problem (Graphical Rewrite Rules):
  −Picture Abstraction: bei der Festlegung einer Regel werden im "vorher"−Teil bei einem Objekt verschiedene Abstraktionen aufgelistet, von diesen muss eine gewählt werden (Bsp.: der graue Stein, ein grauer Stein, irgendein Stein, irgendein Objekt).
  −Property Abstraction: die Eigenschaften können im "vorher"−Teil einer Regel getestet werden. Diese Tests müssen wahr ("true") sein, damit eine Regel durchgeführt wird. Numerische Eigenschaften werden mit ><= getestet.
−Rule−Semantics Problem (Graphical Rewrite Rules):
  −Wird gelöst durch Programming By Demonstration.
−Rule−Sequencing Problem (Graphical Rewrite Rules):
  −Dieses Problem wird bei KidSim nicht gelöst.
−Representation Problem (Programming By Demonstration):
  −KidSim benutzt Graphical Rewrite Rules als Repräsentation der aufgezeichneten Aktionen.

Eine Evaluation von KidSim
Das Ziel von KidSim war es, dass Kinder über das Spielerische implementieren und modifizieren von symbolischen Simulationen die grundlegenden Fähigkeiten zum Programmieren erlernen sollten. Obwohl es weit davon entfernt ist, eine komplette Programmiersprache zu sein, enthält KidSim wichtige Konzepte der Computerprogrammierung (sequentielle Ausführung von Kommandos, einfache Vererbung von Attributen (Variablen) und Regeln, Abstraktion in Regeln) (Cypher & Smith, 1995).

Gilmore et al. fanden 1995, dass Kinder im Alter von 10−13 Jahren sehr schnell lernen, diese Programmierumgebung zu beherrschen. Es macht ihnen Spaß, das System zu benutzen und regt ihre Fantasie an. Es konnte nicht geklärt werden, ob die Kinder wirklich ein tieferes Verständnis der unterliegenden Programmierkonzepte erlernten. Rader et al. fanden 1997 in einer Untersuchung mit 11−14jährigen, dass diese keine wesentliche Aspekte der Programmieroperationen erlernten. Das Fazit dieser Studie war, das KidSim nicht dazu dienen kann, Kindern das Verständnis für Programmierkonzepte zu vermitteln. Sie können allerdings anhand dieses Programms lernen, wissenschaftliche Modelle und Simulationen zu verstehen.

Zusammenfassung
KidSim ist ein Werkzeug, dass es Kindern und Erwachsenen die nicht programmieren können sehr einfach macht, Simulationen zu konstruieren und zu verändern. Es ist ein neuer Ansatz des Programmierens, da er ohne traditionelle Computersprachensyntax auskommt. Aufbauend auf den Erkenntnissen aus der Konstruktion von GUI's, kombiniert KidSim zwei Ideen: Graphical Rewrite Rules und Programming By Demonstration. Das Resultat löst das End User Programming Problem für einige Typen von Simulationen.
Durch die ursprüngliche Einschränkung der Entwickler auf symbolische Simulationen innerhalb einer beschränkten "Spielwelt" ist allerdings auch der Einsatz von KidSim nur begrenzt möglich. Auch als Werkzeug um spielerisch Programmierkonzepte zu erlernen, taugt es nur bedingt.

6. Programmbeispiel: Narval

Das Programm Narval wurde von der Firma Logilab unter der GNU Public License entwickelt. Es ist sowohl für Linux wie für Windows erhältlich. Narval ist die Abkürzung für "Network Assistant Reasoning with Validating Agent Language". Narval ist ein intelligenter, persönlicher Software Assistent, der in der virtuellen Welt lebt und den Anwender unterstützt. Er kann Anwender−definierte Aufgaben durchführen und vorhandene Software und Geräte einsetzen. Er integriert vorhandene Funktionalität und erleichtert deren Ausführung (Fayolle & Cayrol, 2000).
Narval führt gespeicherte Rezepte (recipes) aus. Rezepte sind Sequenzen von Aktionen, die über Transitionen verbunden sind. Diese Transitionen können zur Kontrolle der Ausführung mit Bedingungen verbunden werden. Aktionen, wie beispielsweise das downloaden einer Seite, eine Datenbankabfrage, das Verschicken einer e−Mail, werden als Python−Scripts gespeichert. Fertige Aktionsscripte gibt es in Form von Bibliotheken, sie können auch vom Anwender selbst geschrieben werden. Rezepte werden mit einer graphischen Benutzeroberfläche (Horn) erstellt und verändert. Hier werden die Rezepte als Graphen dargestellt. Aktionen aus einer Bibliothek können hier mit wenigen Mausklicks zu neuen Rezepten zusammengestellt werden (Fayolle & Cayrol, 2000).
Die Designphilosophie von Narval ist "ready for use, easy to use". Narval soll von jedem Anwender bedient werden können, auch von dem mit sehr wenig Computererfahrung. Die Erstellung neuer Rezepte aus fertigen Aktionen mit Horn ist ohne besondere Kenntnisse zu bewerkstelligen. Narval ist ein Agenten−Framework, mit dem man mit wenigen Mausklicks seinen persönlichen, personalisierten Assistenten erschaffen kann.
Narval wird bereits in den verschiedensten Funktionen als persönlicher Software−Assistent eingesetzt: zum Filtern und Beantworten von e−Mails, zum Herausfiltern von Werbung aus Webseiten, zum Zusammenstellen einer Seite mit Nachrichten von verschiedenen Webseiten, zum Versenden von Erinnerungsnachrichten für Verabredungen.
Narval ist ein Produkt das, obwohl es schon einsatzfähig ist, noch am Anfang seiner Entwicklung steht. Für die Zukunft sind noch verschiedene Features geplant, beispielsweise maschinelle Lerntechniken und Kooperation zwischen Agenten.

7. Zusammenfassung

Zur Erschaffung von persönlichen Agenten ist die Programmierung durch den Anwender, im Zusammenspiel mit anderen Techniken (maschinelles Lernen, Wissensbasis) eine wesentliche Methode. Da Anwender in der Regel keine Programmiersprachen beherrschen, wird eine Alternative zur traditionellen Programmierung gesucht. Dabei spielt das Programmieren mit einer graphischen Benutzeroberfläche eine wesentliche Rolle (wichtige Prinzipien dabei: Graphical Rewrite Rules, Programming By Demonstration). Gezeigt wurde die Umsetzung bei der Programmierung von Agenten über graphische Benutzeroberflächen an zwei Beispielen: den Programmen KidSim und Narval.
Abschließend läßt sich sagen, daß die Programmierung von Agenten über graphische Benutzeroberflächen erst teilweise möglich ist. Dabei ist diese Technik im Zusammenspiel mit anderen Anpassungs− und Lerntechniken wichtig für die Erschaffung von personalisierten Agenten.

8. Literatur

−Cypher, A., 1993: Bringing Programming to End Users. In Cypher, A., ed.: Watch what I do: Programming by demonstration. MIT Press, Cambridge, MA, pp. 1−11.
−Cypher, A., Smith, A., 1995: KidSim: End User Programming of Simulations. Proceedings of CHI, ACM, New York, pp. 27−34.
−Fayolle, A., Cayrol, O., 2000: Narval's User Manual. Logilab. URL: http://www.logilab.org/narval/doc/user/en/
−Gilmore, D., Pheasey, K., Underwood, J., Underwood, G., 1995: Learning graphical programming: An evaluation of KidSim. In HCI: Interact '95 Proceedings of fifth IFIP Conference on Human Computer interaction, Lillehammer, Norway.
−Maes, P., 1994: Agents that Reduce Work and Information Overload. Communications of the ACM, 37(7).
−Rader, C., Brand, C., Lewis, C., 1997: Degrees of Comprehension: Children's Understanding of a Visual Programming Environment. CHI 97 Electronic Publications, Atlanta, ACM Press.
−Smith, D.V., Cypher, A., Spohrer, J., 1994: KidSim: Programming Agents without a programming Language. Communications of the ACM, 37(7), pp. 54−67.
−Wagner, D. N., 2000: Software Agents take the Internet as a Shortcut to Enter Society: A Survey of New Actors to Study for Social Theory. First Monday, Vol. 5, Nr. 7.
URL: http://firstmonday.org/issues/issue5_7/wagner/index.html/