AUTOMATYKA • 2009 • Tom 13 • Zeszyt 3

Konrad Kułakowski*, Jarosław Wąs*

Architektura systemu modelowania dynamiki pieszych

1. Wprowadzenie

W ostatnich latach zagadnienie modelowania dynamiki pieszych zyskało duże znaczenie. Jeszcze w latach dziewięćdziesiątych ubiegłego wieku, wraz z pionierskimi pracami Helbinga [8], Fukui oraz Ishibashi [5] czy Blue i Adlera [1], symulacja pieszych była traktowane zaledwie jako ciekawostka. Obecnie, w krajach wysokorozwiniętych, trudno sobie wyobrazić projektowanie obiektu użyteczności publicznej, bez zastosowania odpowiednich narzędzi do symulacji przepływu pieszych. Symulacje te przeprowadza się dla różnych klas obiektów w różnych kontekstach sytuacyjnych: począwszy od standardowego obiektu i codziennego jego użytkowania, skończywszy na symulacji stanu zagrożenia, gdy konieczna jest ewakuacja z budynku. Zastosowane narzędzia informatyczne muszą podlegać dokładnej kalibracji oraz wieloetapowej walidacji, np. przy użyciu normy ISO TR 13387 [10], opisującej m.in. wymagania dotyczące modelowania sytuacji ewakuacyjnych.

W ciągu ostatnich lat autorzy prowadzili szereg prac badawczych i projektowych związanych z dynamiką pieszych począwszy od modeli opartych na prostym automacie komórkowym [18], poprzez bardziej skomplikowane modele niehomogeniczne i asynchroniczne, czy ostatnie modele oparte na systemach agentowych [19].

W chwili obecnej projektowane systemy dynamiki pieszych muszą sprostać coraz większym wymaganiom (coraz większa liczba elementów modelu), przy tym wymaga się od nich bardzo rozbudowanej funkcjonalności. Pomimo ich ciągłego rozwoju daje się zauważyć brak narzędzi umożliwiających symulacje, które są wieloplatformowe i umożliwiają sprawne zaprojektowanie i przeprowadzenie symulacji dynamiki pieszych dla różnych sytuacji. Celem niniejszej pracy jest zaproponowanie architektury takiego systemu oraz zaprezentowanie narzędzia stworzonego w oparciu o przyjęte założenia.

2. Systemy do modelowania tłumu

Można wyróżnić kilka charakterystycznych podejść w modelowaniu tłumu:

– modele ciągłe bazujące na metodzie dynamiki molekularnej; w szczególności stosowane są modyfikacje metody *Social Force*, np. [8],

* Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie

– modele bazujące na niehomogenicznych i asynchronicznych automatach komórkowych, np. [1, 5];
– modele opierające się na systemach wieloagentowych, np. [4].

Modelem, który wywarł wielki wpływ na rozwój modelowania tłumu pieszych, był model *Social Forces* [5]. Model ten bazuje na dobrze zdefiniowanych pojęciach fizycznych dynamiki molekularnej. Piesi są tu reprezentowani w postaci obiektów (ciał), które podlegają działaniu sił oraz oddziałują siłami na innych. Model *Social Forces* był bodajże pierwszym modelem prezentującym mikroskopowe podejście w modelowaniu tłumu, który jest tak szeroko znany i do dziś stosowany w wielu komercyjnych narzędziach.

Z kolei wśród modeli makroskopowych do klasyki należy hydrodynamiczny model Paulsa [14] opierający się na równaniach przepływu. Jednakże podejście makroskopowe sprawdza się w praktyce dla bardzo wąskiej grupy zastosowań.

Na efektywne tworzenie modeli mikroskopowych pozwoliło dopiero zastosowanie niehomogenicznych oraz asynchronicznych automatów komórkowych (CA). Wśród tych pionierskich prac należy wspomnieć model ruchu dwukierunkowego zaproponowany w [1] oraz model unikania kolizji zaproponowany w pracy [5]. W kolejnych pracach podkreślano różne aspekty modelowania oraz rozmaitą metodologię, np. koncepcja statycznych i dynamicznych pól potencjalnych [2] złożonych algorytmów zachowań ludzi. Należy podkreślić tendencję ostatnich lat do rozbudowywania zachowań ludzi, uwzględniania wpływu grup czy rozbudowy funkcjonalności. Skutkuje to rozbudową złożoności i wymagań algorytmów i powstałe w ten sposób modele nie są już właściwie klasyfikowane jako automaty komórkowe, a systemy agentowe [4]. Jednoznaczne przypisanie modelu do grupy automatów komórkowych oraz systemów agentowych jest często bardzo trudne. Zazwyczaj stosuje się kryterium klasyfikacji mówiące, że w systemach opartych na CA piesi wykazują zdolności operacyjne i taktyczne, zaś w podejściu agentowym dodatkowo zdolności strategiczne.

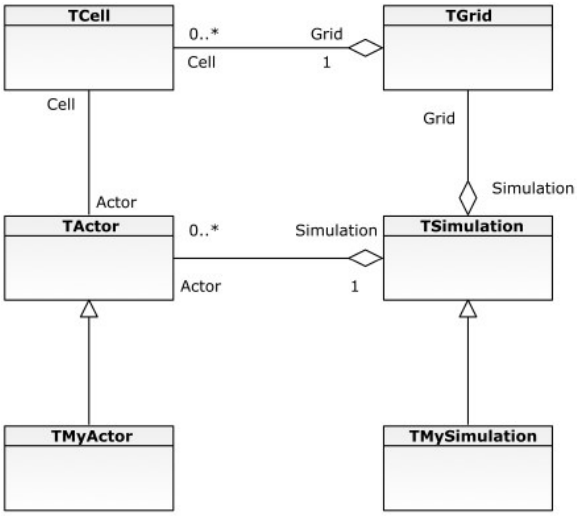
W niniejszej pracy ograniczymy się do dwóch ostatnich podejść w modelowaniu tłumu: automatów komórkowych i systemów wieloagentowych, gdyż te dwa podejścia są od dłuższego czasu przedmiotem badań autorów.

2.1. Istniejące architektury systemów symulacji zachowania pieszych

Istotnym problemem w dyskusji na temat stosowanych architektur w systemach dynamiki pieszych jest brak literatury opisującej wprost to zagadnienie. Większość publikacji związanej z systemami dynamiki pieszych nie opisuje architektury systemu symulacji, lecz w najlepszym wypadku wyjaśnia algorytm sterujący przebiegiem eksperymentu [1, 2, 5, 7].

Jednym z nielicznych wyjątków poruszających tematykę architektury systemu jest praca Dijkstry [4]. W pracy tej przyjęto schemat budowy systemu (rys. 1), w którym dyskretna przestrzeń automatu komórkowego jest opisywana za pomocą klasy *TGrid,* będącej agregacją pewnej ilości komórek reprezentowanych przez *TCell*. Wszystkie możliwe siatki są dostępne w modelu z poziomu klasy *TSimulation*. Podobnie, w klasie *TSimulation*,

zostali zgrupowani agenci reprezentowani przez klasę *TActor*. Uszczegółowieniem klasy *TActor* jest klasa *TMyActor* (opisująca konkretnych agentów w danej symulacji). Odpowiednio uszczegółowieniem klasy *TSimulation* jest klasa *TMySimulation*. Ta ostatnia klasa dostarcza również siatek na potrzeby konkretnych eksperymentów symulacyjnych.



Rys. 1. Diagram klas modelu Dijkstra *et al*. [3]

W systemie Dijkstry [4] jedyna możliwa topologia przestrzeni to dwuwymiarowa siatka. Założenie to jest obecne w ogólnym projekcie systemu (rys. 1).

W prezentowanym w tej pracy podejściu jedynym założeniem architektonicznym dotyczącym otoczenia, w którym poruszają się piesi, jest to, by można było wskazać bezwzględną pozycję każdego obiektu w przestrzeni symulacji (rys. 2). Dopiero każdy konkretny eksperyment doprecyzowywuje przedstawiony generalny schemat, implementując ogólne pojęcia takie jak *Space* czy *Algorithm* (rys. 2). Dzięki temu w ramach prezentowanego rozwiązania można w łatwy i przejrzysty sposób przygotować i przeprowadzić wiele różnych eksperymentów dotyczących zagadnień związanych z dynamiką pieszych, unikając kłopotliwej konstrukcji osobnych aplikacji na potrzeby różnych analiz.

3. Proponowany model systemu

W każdym nieco bardziej złożonym systemie informatycznym kluczową rolę mającą wpływ na jakość tworzonego systemu, jego funkcjonalność oraz późniejszą łatwość pielęgnacji i rozbudowy ma ogólny projekt architektury [12]. Wprowadza on do projektu pojęcia specyficzne dla dziedziny problemowej tworzonego systemu, przekładając je na język zrozumiały dla programisty. Tworzy też podstawy całej aplikacji, wyznaczając podstawowe

struktury danych i miejsca ich przetwarzania oraz podział na moduły. Często też wskazuje technologię, w ramach której projekt zostanie wykonany.

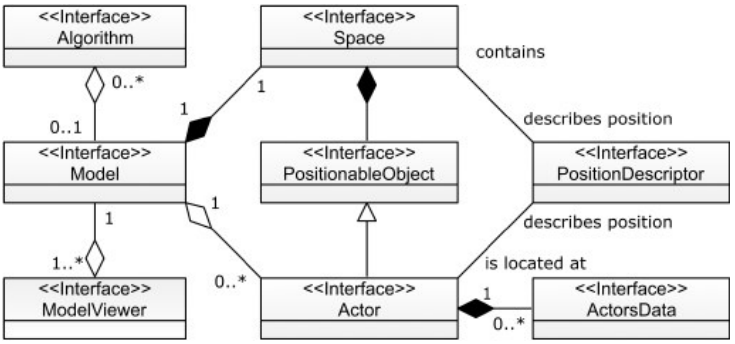
W prezentowanej sekcji został przedstawiony projekt architektury systemu modelującego dynamikę pieszych. Z uwagi na dużą popularność metodologii obiektowej, a co za tym idzie, potencjalnie duże grono odbiorców, autorzy pracy zdecydowali się na wykorzystanie do tego celu diagramów języka UML (*Unified Modelling Language*) [13]. Z przyczyn oczywistych w pracy nie jest prezentowany cały projekt, a tylko kilka wybranych diagramów stanowiących ogólny zarys architektury systemu.

3.1. Zastosowanie platformy Java

Do tworzenia prototypu systemu została wykorzystana platforma JSE 6 (*Java Standard Edition 6*). Java jako język obiektowy dobrze przystaje do metodologii OOD (*Object Oriented Design*) i OOP (*Object Oriented Programming*) ponadto pozwala na testowanie i analizę poważnych i skomplikowanych problemów algorytmicznych związanych z przetwarzaniem współbieżnym, synchronizacją i wydajnością w sposób, który bez uzależniania kodu prototypu od danej platformy sprzętowej. W przekonaniu autorów środowisko Javy pozwala również na szybsze tworzenie niezawodnego kodu aplikacji współbieżnych, tak często będących podstawą eksperymentów symulacyjnych. Warto podkreślić jednak, że w świetle dostępnych opracowań [3, 15] najczęściej porównywany z Javą język C++ wypada dość podobnie i to zarówno w kontekście wydajności obliczeniowej, jak i szybkości tworzenia aplikacji, jakkolwiek prace te nie uwzględniały osobno tworzenia aplikacji wielowątkowych.

3.2. Ogólna architektura

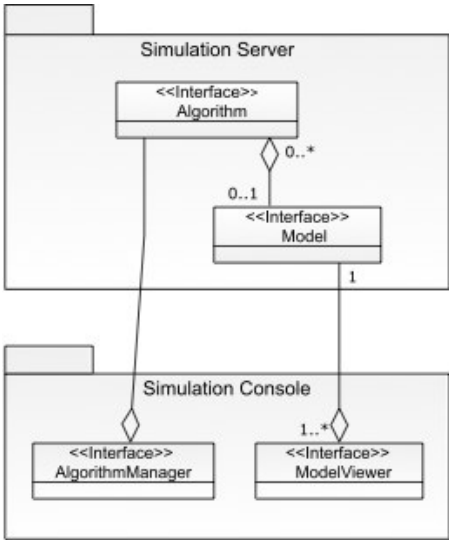
Dwoma najważniejszymi pojęciami w proponowanym modelu architektonicznym systemu modelowania dynamiki pieszych są: *Model* i *Algorithm* (rys. 2). Interfejs *Model* jest agregacją łączącą w sobie pojęcie przestrzeni (*Space*) oraz pojęcie aktora (*Actor*).



Rys. 2. Ogólny schemat architektury systemu

Model może być obserwowany poprzez obiekt wizualizujący (*ModelViewer*). Warstwa prezentacji, jaką jest *ModelViewer* oraz warstwa modelu powinny być logicznie i fizycznie odseparowane za pomocą dodatkowych interfejsów komunikacyjnych. Taki rozdział z jednej strony umożliwia niezależne przetwarzanie modelu, np. przeprowadzanie kosztownych obliczeniowo eksperymentów na wysokowydajnych serwerach aplikacji, z drugiej natomiast pozwala na swobodę w operowaniu aplikacją zapewniającą wizualizację modelu. Pożądane jest, by komunikacja pomiędzy obiektami implementującymi *Model* i *ModelViewer* odbywała się z wykorzystaniem internetu. Interfejs *Algorithm* odpowiada za dynamikę modelu, aktualizację jego stanów i przetwarzanie danych. Implementacja tego interfejsu mieści w sobie szczegóły logiki planowanej symulacji. Takie użycie interfejsu Algorithm jest zgodne z wykorzystaniem znanego wzorca projektowego *Strategy* [6].

Do przeprowadzenia eksperymentu potrzeba zdefiniować i zaimplementować algorytm i model, na którym on operuje. Aby mieć kontrolę nad eksperymentem, potrzeba wiedzieć, jaki jest aktualny stan eksperymentu oraz móc sterować algorytmem. Spostrzeżenie to prowadzi do rozdzielenia wcześniej opisanych interfejsów na dwa komponenty, z których pierwszy zawiera *Model* i *Algorithm* a drugi *ModelViewer* i *AlgorithmManager* (rys. 3).



Rys. 3. Ogólny schemat środowiska symulacji

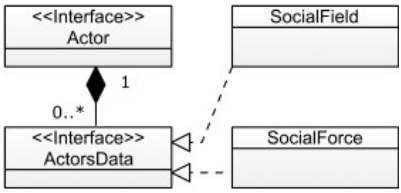
Pierwszy z powstałych komponentów będzie nazywany *SimulationServer*, natomiast drugi – *SimulationConsole*. Na funkcjonalność SimulationConsole składa się ModelViewer odpowiedzialny za wizualizację modelu oraz AlgorithmManager udostępniający metody pozwalające na kontrolę wykonania algorytmu; zatrzymanie i uruchomienie symulacji, zapis stanu symulacji do pliku czy też ręczną modyfikację modelu. W ten sposób zdefiniowa-

ne moduły serwera i konsoli mogą zostać odseparowane od siebie fizycznie i działać na różnych maszynach, komunikując się ze sobą za pomocą internetu. Implementacja takiego rozwiązania wymaga bardzo starannego zaprojektowania protokołu komunikacyjnego pomiędzy modelem a jego wizualizacją. W szczególności niedopuszczalne jest po zmianie stanu modelu każdorazowe przesyłanie wszystkich informacji związanych z modelem, a, tylko tych, które uległy zmianie i są niezbędne dla poprawnej prezentacji modelu.

3.3. Model obiektu pozycjonowalnego

Model jest agregacją obiektów pozycjonowalnych, to znaczy takich, którym można przypisać pozycję, w kontekście pewnej przestrzeni. Pozycja jest reprezentowana przez interfejs *PositionDescriptor*, przestrzeń natomiast jest wprowadzona poprzez interfejs *Space* (rys. 1). To czym jest pozycja w praktyce, jakie rodzaje obiektów pozycjonowalnych przechowuje model zależy od rodzaju i reprezentacji przestrzeni. W przypadku przestrzeni danej n-wymiarowym automatem komórkowym, pozycja będzie opisywana za pomocą n-elementowego wektora współrzędnych, natomiast w przypadku przestrzeni opisanej grafem, pozycja może być zadana etykietą wierzchołka, w którym aktualnie dany obiekt pozycjonowalny się znajduje.

Aktor również jest obiektem pozycjonowalnym. W zależności od rodzaju eksperymentu klasa aktor może posiadać różne implementacje. Na przykład w modelu dynamiki pieszych aktorem będzie obiekt klasy *Pedestrian*, a dla systemu symulującego zachowanie robotów obiekt klasy *Bot* (rys. 4). Z aktorem wygodnie jest czasem związać dodatkowy zestaw informacji opisujący jego stan, kierunek ruchu itp. Wspólnym interfejsem umożliwiającym zunifikowany dostęp do tych danych będzie *ActorsData* (rys. 4). W przypadku analizy ruchu pieszych, w zależności od przyjętego modelu symulacyjnego takimi danymi mogą być obiekty niosące informacje o polach lub też o siłach socjalnych (*social fileds, social forces*).

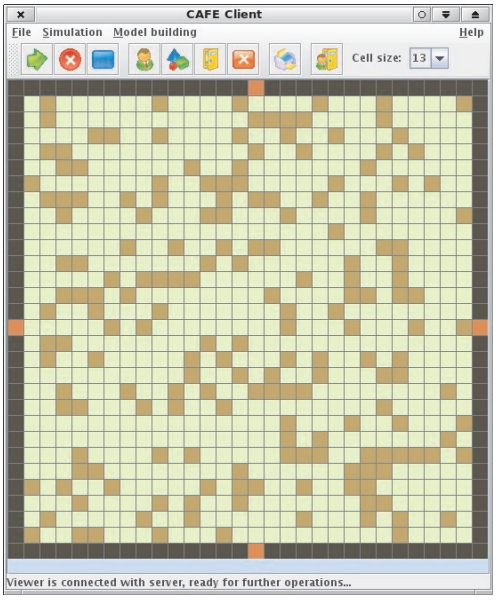


Rys. 4. Model Aktora

3.4. Prototypowa implementacja

Prezentowana architektura została częściowo zaimplementowana w środowisku symulacyjnym *Cafe*. Zgodnie z proponowaną ogólną strukturą rozwiązania (rys. 3) składa się

ono z dwóch modułów: *Cafe Client* – części odpowiedzialnej za wizualizację stanu symulacji i zapewnienie części interfejsu użytkownika umożliwiającej kontrolę przebiegu algorytmu oraz *Cafe Server* – modułu przeprowadzającego symulację. Dzięki obiektowej strukturze obu modułów w łatwy sposób w obrębie jednej aplikacji można umieścić wiele różnych algorytmów symulacji i uruchamiać je w zależności od wybranego typu eksperymentu. W chwili obecnej są prowadzone prace nad dwoma różnymi eksperymentami, z których jeden dotyczy ewakuacji pieszych z pomieszczenia o kilku wyjściach, a drugi, sterowania autonomicznym, mobilnym robotem pilnującym zadanego budynku. W każdym przypadku przeprowadzenie eksperymentu polega na stworzeniu za pomocą edytora (*Cafe Client*) modelu symulowanego otoczenia zawierającego elementy wcześniej zdefiniowanych typów takich jak przeszkody, ściany, wyjścia (punkty atrakcji), piesi, roboty. Następnie można uruchomić symulacje takiego modelu (zgodnie z zadanym algorytmem) oraz obserwować zmiany modelu. Po przeprowadzeniu eksperymentu następuje ewaluacja wyników pod kontem efektywności sterowania (w przypadku robota), realizmu zachowania się ludzi, optymalnej geometrii zadanych ciągów komunikacji pieszej (w przypadku dynamiki pieszych). Środowisko *Cafe* (rys. 5) dostarcza mechanizmów wizualizacji przestrzeni zadanej w postaci różnych rodzajów siatek: złożonej z komórek o kształcie kwadratu, pięciokąta foremnego, mieszanej złożonej z ośmiokątów foremnych i kwadratów. Różne rodzaje komórek oznaczane są różnymi kolorami (np. piesi są reprezentowani przez komórki koloru brązowego). Istnieje możliwość związania z każdą komórką prostego napisu (np. wartości pola potencjału związanego z danym miejscem).



Rys. 5. Pomieszczenie z grupą ludzi – przykładowy model w środowisku *Cafe*

4. Podsumowanie

W artykule przedstawiono koncepcję architektury systemu na potrzeby modelowania i symulacji dynamiki pieszych, a także krótko przedstawiono prototypową implementację takiego systemu (środowisko *Cafe*). W dostępnej literaturze nie często można spotkać wzmiankę o architekturze systemów symulacji dynamiki pieszych. Być może jednym z powodów jest (przynajmniej części z nich) ich komercyjny charakter, a co za tym idzie, brak publicznie dostępnych dokumentów projektowych. Brak opisów architektury dla tej klasy systemów nie oznacza, że nie jest ona ważną składową proponowanej metody czy rozwiązania. Wprost przeciwnie – niejasna struktura aplikacji, której zadaniem było eksperymentalnie potwierdzić założenia i postulaty zgłoszone przez twórców danego algorytmu czy rozwiązania, pozwala wątpić, czy przeprowadzona z pomocą tej aplikacji weryfikacja rozwiązania była poprawna. W tym kontekście przedstawiona praca wydaje się ciekawą odmianą. Prezentowany jest w niej zarys obiektowej architektury aplikacji, która choć podana ogólnie może być z powodzeniem wykorzystana w praktyce przez innych. Czytelne odwołania do wzorów projektowych oraz praktyki projektowania obiektowego powinny pomóc czytelnikom zrozumieć przedstawione założenia architektoniczne. W chwili obecnej gotowa jest prototypowa implementacja środowiska symulacji zgodnego z prezentowaną architekturą. Dwa różne algorytmy symulacji zostały częściowo zaimplementowane i przetestowane. Dalsza rozbudowa środowiska *Cafe* obejmować będzie prace nad zrównolegleniem algorytmów symulacji, weryfikacją zdefiniowanych już modeli i optymalizacją warstwy prezentacji.

Podziękowania

Projekt finansowany ze środków na naukę, jako projekt badawczy własny MNiSW nr.: N N516 228735 oraz ze środków AGH w ramach umowy nr.: 10.10.120.105.

Literatura

[1] Blue V., Adler J., *Bi-dirctional emergent fundamental pedestrian flows from cellular automata microsimulation*. Proceedings of ISTTT'99 Amsterdam, 1999, 235–254.

[2] Burstedde C.K., Klauck K., Schadschneider A., Zittartz J., *Simulation of pedestrian dynamics using a 2-dimensional cellular automaton.* Phys. Rev. A 295, 507–525, 2001.

[3] Comstock C., *Strategic Software Development: Productivity Comparisons of General Deveopment Programs*. World Academy of Science, Engineering and Technology, 34, 2007.

[4] Dijkstra J., Jessurun A. J., Timmermans H., *A multi-agent cellular automata model of pedestrian movement.* Pedestrian and Evacuation Dynamics. Springer-Verlag, Berlin, 2000, 173–181.

[5] Fukui M., Ishibashi Y., *Self-organized phase transitions in CA-models for pedestrians.* J. Phys. Soc. Japan, 1999, 2861–2863.

[6] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[7] Georgoudas I., Sirakoulis G., Andreadis I., *A simulation tool for modeling pedestrian dynamics during evacuation of large areas*. [w:] IFIP Intemational Federation for Information Processing, vol. 204, Artificial Intelligence Applications and Innovations 2006.

[8] Helbing D., Molnar P., *A social force model for pedestrian dynamics.* Phys. Rev. E 51, 1995, 4284–4286.
[9] Lohner R., *On modeling of pedestrian motion*. Applied Mathematical Modeling doi:10.1016/j.apm. 2009.04.017, 2009.
[10] ISO: Norma ISO TR 13387, 1999.
[11] Kułakowski K., Kostrzewa M., *Rapid prototyping of real-time reactive systems.* [w:] International conference on signals and electronic systems (ICSES), 2008.
[12] McConnell S., *Software Project Survival Guide: How to Be Sure Your First Important Project Isn't Your Last*. Microsoft Press, 1997.
[13] Object Management Group (OMG). *Unified Modeling Language 2.0.* OMG http://www.omg.com/uml/", 2005.
[14] Pauls J., *Movement of people.* DiNenno, Washington, 1994.
[15] Prechelt L., *An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl*. http://page.mi.fu-berlin.de/prechelt/Biblio, 2000.
[16] Singh A., Arter R., Louise. D, Langston P., Lester E., Drury J., *Modelling subgroup behavior in crowd dynamics DEM simulation.* Applied Mathematical Modelling 2009 doi:10.1016/j.apm.2009.03.020.
[17] Taubock S., Breitenecker F., *Spatial modeling approaches in DEVS Simulation Systems for Pedestrian Dynamics.* Simulation News Europe, vol. 44–45, 2005.
[18] Wąs J., Gudowski B., *Zastosowanie automatów komórkowych w modelowaniu dynamiki pieszych.* Automatyka (półrocznik AGH), t. 8. z. 3, 2004.
[19] Wąs J., *Multi-agent Frame of Social Distances Model.* ACRI, Lecture Notes in Computer Science, Springer-Verlag, 2008.