(19) OFICIUL DE STAT
PENTRU INVENŢII ŞI MĂRCI
Bucureşti

ROMÂNIA

(11) **RO 128326 A0**
(51) Int.Cl.
G05B 19/418 (2006.01);
G05B 19/042 (2006.01);
G06F 9/00 (2006.01)

(12) # CERERE DE BREVET DE INVENŢIE

(21) Nr. cerere: **a 2012 00700**

(22) Data de depozit: **04.10.2012**

(41) Data publicării cererii:
30.04.2013 BOPI nr. **4/2013**

(71) Solicitant:
• **UNIVERSITATEA "TRANSILVANIA" DIN BRAŞOV**, *BD.EROILOR NR.29, BRAŞOV, BV, RO*

(72) Inventatori:
• **ITU ALINA**, *STR. MESTECĂNIŞ NR. 3, BL. A35B, SC. B, AP. 24, BRAŞOV, BV, RO;*
• **SISAK FRANCISC**, *STR. 13 DECEMBRIE NR. 57, BL. 23, SC. B, AP. 8, BRAŞOV, BV, RO;*
• **SUCIU CONSTANTIN**, *STR. TRAIAN VUIA NR. 1, SC. A, AP. 3, BRAŞOV, BV, RO*

(54) # ARHITECTURĂ PENTRU OPTIMIZAREA PROCESELOR DE FABRICAŢIE DIN CADRUL ÎNTREPRINDERILOR

(57) **Rezumat:**

Invenţia se referă la o arhitectură pentru optimizarea proceselor de fabricaţie din cadrul întreprinderilor, atât din punct de vedere al timpului de fabricaţie, cât şi din punct de vedere al timpului de instalare. Arhitectura conform invenţiei este compusă din trei componente principale: prima componentă este un server OPC UA (OLE for Process Control Unified Architecture - Arhitectură Unificată OPC) care modelează o serie de date de la nivelul dispozitivelor utilizate în procesul de fabricaţie, iar pentru construirea eficientă a spaţiului de adrese, a fost proiectat un set de algoritmi care generează automat nodurile UA ale spaţiului de adrese, pe baza conţinutului unei surse de date; a doua componentă este reprezentată de o serie de servicii software organizate pe două niveluri, cu rol de asigurare a unei comunicaţii mai uşoare între nivelul superior CSP (Constraint Satisfaction Problems - Probleme de Satisfacere a Constrângerilor) şi nivelul inferior OPC UA, aceste servicii fiind introduse la cel de-al doilea nivel, deoarece garantează flexibilitatea şi reutilizarea întregii arhitecturi; iar a treia componentă este reprezentată de un set de modele CSP care determină soluţiile de producţie optimizate, şi care sunt folosite automat, prin intermediul arhitecturii.

Revendicări: 5
Figuri: 13

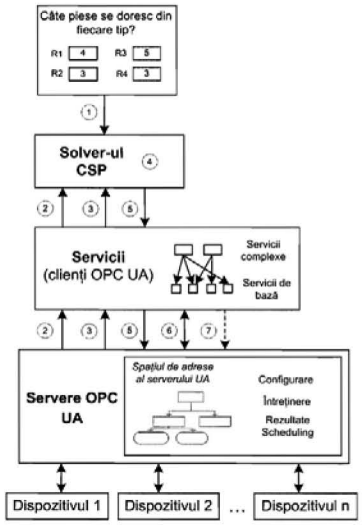

Fig. 2

RO 128326 A0

Nr. cat. DPI: 172/20.09.12

135

ARHITECTURĂ PENTRU OPTIMIZAREA PROCESELOR DE FABRICATIE DIN CADRUL ÎNTREPRINDERILOR

Invenția se referă la o arhitectură pentru optimizarea procesului de fabricare în întreprinderi prin determinarea unor planuri şi orare de lucru optimizate şi apoi prin utilizarea automată a rezultatelor acestor planuri. La proiectarea arhitecturii au fost folosite trei tehnologii de actualitate: OPC UA (OPC Unified Architecure), SOA (Service Oriented Architecture), CSP (Constraint Satisfaction Problems). Soluția tehnică a fost dezvoltată în special pentru liniile de fabricație flexibile, dar poate fi aplicată pentru orice proces industrial care necesită luarea unor decizii pentru desfăşurarea optimă a acestuia din punct de vedere al timpului de procesare.

Tehnologia OPC UA a fost introdusă în cadrul arhitecturii în principal pentru a permite utilizarea automată, fără intervenția unui operator uman, a planurilor de fabricație optime. În plus, această tehnologie contribuie şi la flexibilitatea şi reutilizabilitatea arhitecturii prin structura spațiilor de adrese (care facilitează flexibilitatea modelelor CSP de la nivelul superior al arhitecturii) şi prin simplitatea definirii acestor spații de adrese (asigurată în principal de algoritmii de generare automată a spațiului de adrese).

În mediul economic actual, companiile trebuie să reacționeze rapid la modificările apărute în cerințele de pe piață şi să respecte termenele limită stabilite de clienți.

Prin urmare specialiştii au ajuns la concluzia că sistemele de fabricație trebuie să fie bazate şi orientate pe timp iar întreaga strategie de fabricație trebuie adaptată pentru a suporta inovarea şi introducerea uşoară de noi produse. În general, pentru a face față acestui nou context, este necesară o mai mare adaptabilitate şi flexibilitate.

Principalele cerințe care trebuie satisfăcute de către întreprinderile de producție şi de sistemele de control includ următoarele [Jammes, F., Smit, H. *"Service oriented paradigms in industrial automation"*, IEEE Transaction on Industrial Informatics, Feb 2005, pp. 62-70.]:

- Capacități de integrare dinamică a entităților în interiorul întreprinderii;
- Cooperare între întreprinderi;
- Suport pentru mediile hardware şi software eterogene şi inter-operabile;
- Agilitate prin adaptabilitate şi reconfigurare;
- Scalabilitate prin adăugare de resurse fără a întrerupe operațiile;
- Toleranță la eroare şi revenire rapidă de la eşecuri.

S-a concluzionat că, în economia globală de azi, pentru a menține competitivitatea și productivitatea, optimizarea proceselor de fabricație este chiar mai importantă decât în trecut [Bohn, H., Bobek, A., Golatowski, F. "SIRENA - service infrastructure for real-time embedded networked devices: a service oriented framework for different domains", International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, Morne, Mauritius, Apr. 2006, pp. 43-47.]. Statisticile curente arată că o treime din costul total de fabricație al unei companii este dat de activitățile de instalare și configurare a echipamentelor [Jammes, F., Smit, H. *"Service-Oriented Architectures for Devices - the SIRENA View"*, Proc. of the 3rd Inter. Conf. on Industrial Automation, Aug. 2005, pp. 140–147.].

Pe de altă parte internetul și tehnologiile web devin din ce în ce mai mult elementele centrale ale unei lumi în care dispozitivele sunt interconectate în diferite moduri. În literatura de specialitate au fost identificate o serie de cerințe și provocări referitoare la ecosistemele de interconectare a dispozitivelor [Jammes, F., Smit, H. *"Service oriented paradigms in industrial automation"*, IEEE Transaction on Industrial Informatics, Feb 2005, pp. 62-70.]:

- Standardele web trebuie folosite de câte ori este posibil;
- Dispozitivele trebuie să furnizeze interfețe universale, interoperabile și cu acces securizat;
- Dispozitivele trebuie să poată fi integrate ușor în cadrul sistemelor complexe și integrarea trebuie să fie scalabilă;
- Fiecare subsistem trebuie să fie expus precum un singur dispozitiv astfel încât să poată fi integrat în sisteme mult mai complexe;
- Interacțiunile dintre dispozitive trebuie să fie previzibile.

În continuare sunt prezentate **progresele recente în aplicarea conceptului SOA** (arhitecturi orientate pe servicii) în producție deoarece arhitectura propusă operează la nivelul MES al piramidei de automatizare, unde se realizează monitorizarea și controlul producției.

Sunt cunoscute diverse metode propuse în proiecte de cercetare finanțate de Uniunea Europeană care au avut ca scop aplicarea conceptului SOA direct la nivelul dispozitivelor (SIRENA, SOCRADES) sau la nivelul ERP al piramidei de automatizare (GRIA). Pe lângă acestea, diverse activități de cercetare au fost direcționate către utilizarea serviciilor web la nivelul dispozitivelor, fie că acestea sunt automate programabile sau calculatoare de proces (precum arhitecturile TiCS, UPnP, DSSP). Se dorește astfel obținerea unei flexibilități sporite la nivelul dispozitivelor care să crească agilitatea și viteza de implementare a modificărilor în producție (de exemplu protocolul IMNP [Gilart-Iglesias, V., Macia-Perez, F., Capella-D'alton, A., Gil-Martinez-Abarca, J. A., *"Industrial Machines as a Service: A*

Model Based on Embedded Devices and Web Services", Proc. of the IEEE Inter. Conf. on Industrial Informatics, Singapore, Singapore, Aug. 2006, pp. 630–635.]). De asemenea există o serie de limbaje de programare pentru descrierea formală a comportamentului aplicațiilor critice din punctul de vedere al securității şi al timpului de execuție: Timber şi Hume. Acestea nu sunt însă utilizate în mod uzual la implementarea arhitecturilor orientate pe servicii.

Analizând toate aceste aspecte **se poate identifica unul din dezavantajele fundamentale** ale migrării către o arhitectură orientată pe servicii: integrarea dispozitivelor existente. Direcțiile care trebuie urmate la crearea unei astfel de arhitecturi au fost stabilite în mai multe lucrări, dar încorporarea echipamentelor existente în noile arhitecturi rămâne o problemă dificilă.

După cum a fost prezentat în lucrarea [Filho, F. Sd. L., da Fonseca, A. L. T. B., de Souza, A. J., Couto, F.A., dos Santos, R. P. R., Guedes, L. A. *"Industrial processes supervision using service oriented architecture"*, 32nd IEEE Conf. Industrial Electronics, Paris, France, Nov. 2006, pp. 4552-56.], unul din motivele principale de a utiliza serviciile web direct la nivelul dispozitivelor este acela că OPC-ul clasic a fost foarte rigid şi dependent de platformă şi tehnologie. Aceste limitări au fost însă eliminate odată cu apariția noii specificații OPC UA, care este susținută de principalele companii în domeniul automatizărilor industriale (Siemens, ABB, Schneider, etc.).

Un alt aspect care trebuie abordat îl reprezintă întârzierile introduse de utilizarea serviciilor web direct la nivelul dispozitivelor. În [Cucinotta, T., Mancina, A., Anastasi, G.F., Lipari, G., Mangeruca, L., Checcozzo, R., Rusina, F. *"A real-time service-oriented architecture for industrial automation"*, IEEE Trans. on Industrial Informatics, Vol. 5, 2009, pp. 257-266.] se realizează o analiză a calității serviciilor în contextul comunicării în timp real. Se evaluează astfel comportamentul temporal al serviciilor pentru a evita interferențele nedorite dintre servicii. **Se poate astfel identifica un dezavantaj** al utilizării serviciilor web la nivelul dispozitivelor, şi anume apariția unor întârzieri nedorite.

În plus există o serie de alte framework-uri de servicii (pe lângă serviciile web) care pot conduce la timpi de execuție superiori, precum arhitectura Apache River, care se bazează pe servicii Java.

În continuare sunt **prezentate progresele recente realizate în domeniul problemelor de satisfacere a constrângerilor** în ceea ce priveşte aplicarea paradigmei CSP în mediul industrial.O prezentare detaliată a meta-euristicilor utilizate în problemele de determinare a orarelor de lucru utilizate în industrie este realizată în [Xhafa, F., Abraham, A. *"Metaheuristics for Scheduling in Industrial and Manufacturing Applications"*, Springer, Berlin, Germania, 2008.]. Modelele MIP care

se pot aplica pentru diverse tipuri de aplicații au fost introduse în [Sawik, T. "*Scheduling in supply chains using mixed integer programming*", Wiley, Hoboken, NY, USA, 2011.].

Este cunoscut un algoritm memetic pentru problemele de tip job-shop [Hansmann, K. W., Hoeck, M., *"Production Control of a Flexible Manufacturing System in a Job Shop Environment"*, International Transactions in Operational Research, Vol. 4, 1997, pp. 341-351.], care sunt cunoscute a fi unele dintre cele mai complicate probleme de combinatorică. Acesta reprezintă practic un algoritm hibrid care combină tehnici de căutare globală şi locală [Gao, L., Zhang, G., Zhang, L., Li, X. "*An efficient memetic algorithm for solving the job shop scheduling problem*", Computers & Industrial Engineering, Vol. 60, 2011, pp. 699-705.]. Noutatea constă în modificarea sistematică a locației de căutare pentru a evita soluții optime local. În acelaşi domeniu, s-a introdus o modalitate mai eficientă de minimizare a condiției de *makespan*, în condițiile în care maşinile pentru care urmează să se stabilească orarul de lucru nu sunt disponibile pe întreaga durată a planificării [Mati, Y. "*Minimizing the makespan in the non-preemptive job-shop scheduling with limited machine availability*", Computers & Industrial Engineering, Vol. 59, 2010, pp. 537-543.]. Tot pentru minimizarea condiției de *makespan*, în [Zribi, N. Ã., Kamel, A. E., Borne, P. *"Minimizing the makespan for the MPM job-shop with availability constraints"*, International Journal of Production Economics, Vol. 112, 2008, pp. 151-160.]se introduc atât o euristică bazată pe reguli prioritare cât şi un algoritm genetic pentru rezolvarea problemei de secvențiere.

În condițiile în care paradigma CSP a fost aplicată pentru probleme de complexitate tot mai mare şi timpii de căutare au crescut exponențial, o parte importantă a activităților de cercetare a fost îndreptată înspre reducerea complexității modelelor CSP [Medland, A.J., Matthews, J. *"The implementation of a direct search approach for the resolution of complex and changing rule-based problems"*, Engineering with Computers, Vol. 27, 2011, pp. 105-115.]. O altă metodologie bazată pe constrângeri, care este aplicată în cazul modelelor complexe a fost introdusă în [Edmunds, R., Feldman, J. A., Hicks, B. J., Mullineux, G. "*Constraint -based modelling and optimization to support the design of complex multi-domain engineering problems*", Engineering with Computers, Vol. 27, 2011, pp. 319-336.], permițând modelarea şi rezolvarea unor sisteme largi cu peste 100 de grade de libertate printr-o combinație de algoritmi direcți sau de evoluție. Tehnica poate fi aplicată în diverse domenii: maşini industriale, modelare umană, modelare geologică, etc.

Un aspect foarte important în mediul industrial este luarea de decizii în timp-real (în timpul funcționării instalațiilor). Un sistem de determinare a orarelor de lucru în timp real a fost introdus în [Frantzen, M., Ng, H. C., Moore, P. R. *"A simulation-based scheduling system for real-time*

optimisation and decision making support", Robotics and Computer Integrated Manufacturing, Vol. 27, 2011, pp. 696-705.], care a fost aplicat în uzina unui producător auto din Suedia. Acest sistem determină orare de lucru în timp real şi transmite sugestii către operatorii umani. Modelele sunt robuste şi permit o reconfigurare rapidă în cazul în care sistemul modelat se modifică.

Un domeniu foarte important pentru aplicarea paradigmei CSP în domeniul industrial este acela al sistemelor flexibile de fabricaţie (flexibilitatea acestora permite o reconfigurare rapidă a producţiei) [Brevet US6039168: Claude, D. "Method of manufacturing a product from a workpiece", 2000.]. Un model axat pe monitorizare este prezentat în [Finkelstein L., 2001] Finkelstein, L., Markovitch, S. *"Optimal schedules for monitoring anytime algorithms"*, Artificial Intelligence, Vol. 126, 2001, pp. 63-108.], în care un set de interogări sunt transmise procesului monitorizat pentru a detecta satisfacerea unui ţel. Aceste interogări consumă timp de la procesul monitorizat, ceea ce duce la o întârziere a momentului de timp la care este îndeplinită condiţia obiectivului. În acest sens este prezentat un model formal pentru aceste tipuri de probleme, se realizează o analiză teoretică a claselor de orare de lucru optime şi se introduce un algoritm pentru construirea unor orare optime de monitorizare.

Într-una din primele lucrări publicate în acest domeniu a fost prezentată aplicarea paradigmei CSP într-un sistem de fabricaţie flexibil pentru haine [Tomastik, R. N., Luh, P. B., Liu, G. "*Scheduling flexible manufacturing systems for apparel production*", IEEE Transactions on Robotics and Automation, Vol. 12, 1996, pp. 789-799.]. Modelul introdus rezolvă o problemă de planificare prin care se decide câte şi care maşini să fie alocate fiecărei sarcini.

În [Zeballos, L. J. *"A constraint programming approach to tool allocation and production scheduling in flexible manufacturing systems",* Robotics and Computer Integrated Manufacturing, Vol. 26, 2010, pp. 725-743.], [Zeballos, L. J., Quiroga, O. D., Henning, G. P. *"A constraint programming model for the scheduling of flexible manufacturing systems with machine and tool limitations"*, Engineering Applications of Artificial Intelligence, Vol. 23, 2010, pp. 229-248.] se introduc atât un model cât şi o strategie de căutare pentru planificarea producţiei unei linii flexibile. Modelul ia în considerare o serie de aspecte esenţiale ale liniilor flexibile: numărul uneltelor, durata de viaţă a acestora, capacitatea magaziilor de unelte, planificarea maşinilor şi a uneltelor, direcţionarea componentelor, etc. O altă activitate de cercetare s-a concentrat în special asupra problemei de încărcare a staţiilor dintr-un sistem de fabricaţie flexibil [Abazari, A.M., Solimanpur, M., Sattari, H. "Optimum loading of machines in a flexible manufacturing system using a mixed-integer linear mathematical programming model and genetic algorithm", Computers & Industrial Engineering, Vol. 62, 2012, pp. 469-478.]. Lucrarea

propune un model matematic liniar compus atât din variabile întregi cât şi din variabile booleene, care este apoi rezolvat prin aplicarea unui algoritm genetic, care conduce la timpi de soluționare mai mici. În [Sawik, T. *"Loading and scheduling of a flexible assembly system by mixed integer programming"*, European Journal of Operational Research, Vol. 154, 2004, pp. 1-19.], se prezintă un model MIP general utilizat pentru a determina simultan soluții de încărcare şi de planificare a stațiilor unei linii flexibile (fiecare stație este compusă din mai multe maşini paralele). Un aspect important neglijat în această lucrare îl reprezintă timpii de instalare ai maşinilor (timpul necesar ca o stație să comute de la un tip de operație la altul). Această problemă este adresată în [Andrés, C., Miralles, C., Pastor, R. *"Balancing and scheduling tasks in assembly lines with sequence-dependent setup times"*, European Journal of Operational Research, Vol. 187, 2008, pp. 1212-1223.], iar modelul MIP introdus anterior în [Sawik, T. *"Loading and scheduling of a flexible assembly system by mixed integer programming"*, European Journal of Operational Research, Vol. 154, 2004, pp. 1-19.] a fost extins prin introducerea unor timpi de instalare dependenți de secvența de execuție [Ozturk, C., Tunali, S., Hnich, B., Ornek, A. M. *"Simultaneous Balancing and Scheduling of Flexible Mixed Model Assembly Lines with Sequence-Dependent Setup Times"*, Electronic Notes in Discrete Mathematics, Vol. 36, 2010, pp. 65-72.]. Lucrarea [Magnusson, P., Sundström, N., Bengtsson, K., Lennartson, B., Falkman, P., Fabian, M. *"Planning transport sequences for flexible manufacturing systems"*, Proceedings of the 18th IFAC World Congress, Milano, Italy, Aug. 2011, pp. 9494-9499.] în schimb se concentrează asupra activităților de transport dintre stațiile unui sistem de fabricație. Se introduce o metodă care identifică, creează şi vizualizează aceste activități pe baza datelor de intrare furnizate de un program virtual de fabricație. O problemă axată exclusiv pe planificarea activităților a fost introdusă în [Özcan, U., Çerçioğlu, H., Gökçen, H., Toklu, B. *"Balancing and sequencing of parallel mixed-model assembly lines"*, International Journal of Production Research, Vol. 48, 2009, pp. 5089-5113.], scopul fiind acela de a planifica simultan mai multe linii de asamblare în contextul unor resurse limitate şi de a distribui în mod echilibrat volumul total de lucru.

Numeroase activități de cercetare s-au concentrat în special asupra modelelor MIP. În [Mosadegh, H., Zandieh, M., Fatemi Ghomi, S. M. T. "Simultaneous solving of balancing and sequencing problems with station-dependent assembly times for mixed-model assembly lines", Applied Soft Computing, Vol. 12, 2012, pp. 1359-1370.] se adresează probleme de planificare şi secvențiere, scopul final fiind acela de minimiza cantitatea totală de lucru. [Unlu, Y., Mason, S. J. *"Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems"*, Computers & Industrial Engineering, Vol. 58, 2010, pp. 785-800.] evaluează patru formulări diferite ale modelelor

MIP cu aplicabilitate în problemele de planificare cu maşini paralele. Într-o altă lucrare, [Dolgui, A., Guschinsky, N., Levin, G. "Enhanced mixed integer programming model for a transfer line design problem", Computers & Industrial Engineering, Vol. 62, 2012, pp. 570-578.], se descrie utilizarea unui model MIP în cazul unei instalaţii industriale multi-ax (multi-şpindlu). Modelul determină şpindlul care trebuie utilizat pentru o anumită operaţie la un anumit moment dat, scopul fiind acela de a minimiza costul total. Un alt model este prezentat în [Westerlund, J., Hästbacka, M., Forssell, S., Westerlund, T. *"Mixed-Time Mixed-Integer Linear Programming Scheduling Model"*, Industrial & Engineering Chemistry Research, Vol. 46, 2007, pp. 2781-2796.], unde este adresată în special problema de depozitare a produselor între staţiile unei linii de producţie complexe. Modelul propus este o combinaţie între modele cu timp continuu şi timp discret şi poate gestiona atât probleme cu durata de lucru scurtă cât şi modele periodice. Astfel se combină flexibilitatea şi eficienţa modelelor de timp continuu cu grid-ul temporal de referinţă, care este utilizat în reprezentările cu timp discret.
Modelele CSP menţionate mai sus adresează aspecte ale nivelului MES al piramidei de automatizare. Paradigma CSP poate însă fi aplicată cu succes şi la nivelul ERP al unui întreprinderi, după cum este prezentat în lucrările [Nieuwenhuyse, I. V., Boeck, L. D., Lambrecht, M., Vandaele, N. J. "Advanced resource planning as a decision support module for ERP", Computers in Industry, Vol. 62, 2011, pp. 1-8.] şi [Hvolby, H.-H., Steger-Jensen, K. *"Technical and industrial issues of Advanced Planning and Scheduling (APS) systems"*, Computers in Industry, Vol. 61, 2010, pp. 845-851.].

În ceea ce priveşte găsirea unei soluţii optime a unui model CSP, în [Duffany, J. L. *"Optimal Solution of Constraint Satisfaction Problems"*, Engineering and Technology, Vol. 37, 2009, pp. 1369-1373.] s-a introdus un algoritm nou bazat pe o tehnică de substituţie similară celei utilizate la rezolvarea sistemelor mari de ecuaţii. Ineficienţa iniţială a acestuia, dată de necesitatea calculului pătratului matricei A la fiecare pas, este eliminată ulterior printr-o tehnică de incrementare succesivă.

Lucrările menţionate mai sus pot fi împărţite în mai multe categorii, precum: planificare pentru probleme de tip job-shop, rezolvarea modelelor cu grad înalt de complexitate, probleme de de determinare a orarelor de lucru în timp real sau pentru sistemele flexibile de fabricaţie, modele MIP, modele pentru nivelul ERP al piramidei de automatizare, etc.

Toate aspectele prezentate indică faptul că domeniul CSP este un domeniu de cercetare care a ajuns la maturitate şi care poate fi aplicat cu succes în rezolvarea problemelor practice. Progresele menţionate sunt remarcabile şi au permis rezolvarea unor modele de complexitate tot mai mare în intervale de timp mai scurte.

Un aspect important care a fost însă neglijat aproape în totalitate în lucrările menționate îl reprezintă utilizarea automată în producție a soluțiilor problemelor CSP. În cazul în care nu există o legătură directă între soluțiile problemelor CSP și dispozitivele de control ale mașinilor industriale, necesitatea unui operator uman care să realizeze această sarcină duce la scăderea semnificativă a eficienței unei astfel de abordări. În plus, mediul industrial este unul dinamic, în interiorul căruia pot mereu să apară modificări datorită schimbării produselor fabricate, a activităților de mentenanță sau a unor defecte ale instalațiilor. Astfel conexiunea dintre un solver CSP și dispozitivele de control, cărora le sunt destinate soluțiile modelelor CSP, nu este necesară doar pentru a utiliza automat aceste soluții, dar și pentru a realiza o reconfigurare continuă și dinamică a modelelor CSP. Eficiența și eleganța paradigmei CSP poate fi aplicată cu succes în mediul industrial doar în cazul în care este stabilită această conexiune bidirecțională.

În lucrarea [Frantzen, M., Ng, H. C., Moore, P. R. *"A simulation-based scheduling system for real-time optimisation and decision making support"*, Robotics and Computer Integrated Manufacturing, Vol. 27, 2011, pp. 696-705.] a fost sugerată o astfel de abordare dar soluțiile problemei CSP erau trimise ca sugestii către operatorii umani și nu direct către dispozitivele de control. Singura lucrare în domeniu care propune o abordare bazată pe utilizarea automată a soluțiilor este [Ferrolho, A., Crisostomo, M. *"Intelligent control and integration software for flexible manufacturing cells"*, IEEE Trans. on Industrial Informatics, Vol. 3, 2007, pp. 3-11.], unde orarele de lucru optim sunt determinate prin algoritmi genetici. **Dezavantajele pe care le are soluția propusă** sunt: capabilități de modelare și flexibilitate reduse precum și integrare dificilă cu nivelul ERP.

Un alt aspect neglijat în lucrările din domeniu îl reprezintă combinația dintre nivelul MES și nivelul ERP al piramidei de automatizare. Prin realizarea unei conexiuni directe dintre dispozitivele de control și modelele CSP utilizate pentru determinarea unor orare de lucru optime, se deschid și o serie de oportunități pentru nivelul ERP: se pot determina valorile indicatorilor cheie de performanță și se pot construi modele CSP la nivelul ERP care să interacționeze cu modelele CSP de la nivelul MES.

Prin urmare, dezavantajele abordărilor propuse până în prezent sunt următoarele:

- lipsa posibilității de integrare a echipamentelor existente în noile tehnologii/arhitecturi;
- introducere unor întârzieri semnificative prin utilizarea serviciilor web în aplicații industriale;
- lipsa posibilității de utilizare automată a soluțiilor modelelor CSP;
- flexibilitate scăzută sau inexistentă;
- capacitate redusă de utilizare la nivelul ERP al piramidei de automatizare a datelor de la nivelul MES.

Scopul invenției a fost de a construi o arhitectură orientată pe servicii care să optimizeze aplicațiile industriale din mai multe puncte de vedere. Aceasta trebuie se permită aducerea mai aproape de proces a deciziilor luate la nivelele superioare ale piramidei de automatizare şi să poată fi aplicată într-o gamă largă de procese industriale.

În acest sens au fost identificate **obiectivele principale la dezvoltarea invenției**:

- optimizarea proceselor de fabricație prin determinarea unor planuri de fabricație optime;
- utilizarea automată, fără intervenția unui operator uman, a planurilor de fabricație optime;
- construirea unei arhitecturi flexibile, adaptabile şi reutilizabile, care să reducă timpii de instalare şi setare la valori minime.

Pentru determinarea unor planuri de fabricație optime se propune metodologia bazată pe probleme de satisfacere a constrângerilor (CSP – Constraint Satisfaction Problems) deoarece pe de o parte permite construirea unor modele flexibile, liniare şi neliniare, iar pe de altă parte există o serie de solvere open source bine dezvoltate care pot fi folosite pentru optimizarea aplicațiilor industriale. În continuare, pentru a permite utilizarea automată a planurilor de fabricație, soluția obținută trebuie comunicată dispozitivelor de control. În acest sens a fost aleasă specificația OPC Unified Architecture care a fost introdusă în 2006 ca înlocuitor al vechiului standard OPC bazat pe tehnologia COM. Standardul OPC clasic a fost creat ca interfață de comunicare dintre drivere, permițând citirea standardizată cât şi acces de scriere asupra datelor curente din dispozitivele de automatizare. Astfel caracteristicile necesare îndeplinirii celui de-al doilea obiectiv al arhitecturii prezentate în continuare erau deja prezente în specificația clasică, deoarece aceasta permitea un acces standardizat, independent de producător, la dispozitivele de control. Vechea specificație avea însă o serie de dezavantaje, dintre care cel mai important era dependența de platforma Windows, dată de utilizarea tehnologiei COM introdusă de Microsoft. Noua specificație OPC UA a înlăturat acest dezavantaj şi a introdus o serie de alte avantaje [Mahnke, W., Leitner, S.H., Damm, M. *"OPC Unified Architecture"*, Springer Press, Berlin, Germany, 2009].

În final, pentru atingerea celui de-al treilea obiectiv s-a ales folosirea serviciilor software. Introducerea logicii în cadrul unor servicii software permite construirea unor aplicații flexibile şi mai ales reutilizarea codului. Serviciile software sunt utilizate în principal pentru a facilita comunicarea dintre serverele OPC UA şi solver-ele utilizate pentru determinarea soluțiilor optime. Prin introducerea serviciilor, cele două nivele pot fi programate şi întreținute independent. Flexibilitatea şi reutilizabilitatea componentelor software nu este dată doar de prezența serviciilor ci şi de modul de realizare şi implementare al nivelului bazat pe constrângeri (la acest nivel modelele definite trebuie să

fie flexibile pentru a ține cont de starea actuală a dispozitivelor şi stațiilor de fabricație) şi al nivelului serverelor OPC UA (aceste servere modelează dispozitivele de control şi modelele astfel obținute, numite în continuare spații de adrese, trebuie şi ele să fie construite în mod eficient dar şi să permită o întreținere uşoară).

Se dă în continuare un exemplu de realizare a invenției împreună cu figurile 1 - 13 care reprezintă:

Fig. 1 - Contribuția tehnologiilor la îndeplinirea obiectivelor arhitecturii/invenției.

Fig. 2 - Arhitectura propusă pentru optimizarea proceselor de fabricație.

Fig. 3 - Etapele de creare a unui server UA.

Fig. 4 - Tipuri de noduri.

Fig. 5 - Nod metodă.

Fig. 6 - Mecanism de rotație.

Fig. 7 - Structura aplicației bazate pe linia flexibilă FMS 200.

Fig. 8 - Operații executate pentru fabricarea pieselor.

Fig. 9 - Spațiul de adrese corespunzător aplicației.

Fig. 10 - Serviciul complex utilizat pentru monitorizarea şi controlul procesului de fabricare a mecanismelor de rotație.

Fig. 11 - Interfața grafică utilizată pentru plasarea unei comenzi.

Fig. 12 - Orar de lucru pentru o comandă de 9 piese.

Fig. 13 - Monitorizarea alarmelor şi a evenimentelor prin intermediul serviciilor.

În continuare este prezentată arhitectura care reprezintă obiectul invenției. **Figura 1** prezintă cele trei obiective ale arhitecturii propuse precum şi tehnologiile care contribuie la îndeplinirea fiecărui scop. Se poate astfel observa cum fiecare tehnologie îndeplineşte în principiu un anumit obiectiv, dar toate cele trei tehnologii contribuie la asigurarea flexibilității arhitecturii.

Arhitectura propusă ca invenție este construită în jurul componentelor cheie SOA (Arhitectură Orientată pe Servicii), ale cărei caracteristici principale sunt: autonomia şi interoperabilitatea. Lucrarea [Jammes, F., Smit, H. *"Service oriented paradigms in industrial automation"*, IEEE Transaction on Industrial Informatics, Feb 2005, pp. 62-70.] a identificat principiile care permit combinarea ambelor proprietăți, dintre care cele mai importante sunt:

- interfața proiectată printr-o abordare din exterior înspre interior;
- implementarea serviciilor să poată fi modificată în orice moment fără a afecta interfața sau alte servicii.

Figura 2 prezintă designul conceptual al arhitecturii, care este compusă din trei nivele principale, reprezentând cele trei tehnologii ilustrate în figura 1. Serverul OPC UA colectează datele de la dispozitive, senzori sau elemente de execuție, le modelează într-un mod unificat şi standardizat şi de asemenea asigură comunicația în timp real cu dispozitivele de control. Cea de-a doua componentă este reprezentată de mai multe nivele de servicii, care reprezintă stâlpii de susținere ai arhitecturii şi care prin urmare garantează flexibilitatea şi adaptabilitatea sa. Cea de-a treia componentă a arhitecturii (solver-ul CSP) adresează scopul principal al acesteia şi anume optimizarea planurilor de producție şi a orarelor de lucru. Modelele de satisfacere a constrângerilor au fost identificate ca fiind un mod simplu şi eficient de furnizare a acestui tip de optimizare. În locul abordării CSP poate fi utilizată însă orice altă tehnologie care permite determinarea unor soluții de fabricare. Astfel, prin combinarea acestor trei concepte, arhitectura optimizează atât producția cât şi activitățile de instalare şi de setare. Dezvoltarea arhitecturii a fost bazată pe convingerea că trebuie urmată o cale de mijloc care să mențină rolul actual al automatelor programabile şi al dispozitivelor de control similare, dar care pe de altă parte îndeplineşte noile cerințe ale mediilor de fabricație.

Figura 2 ilustrează interacțiunile conceptuale dintre cele trei nivele ale arhitecturii. Se porneşte de la ideea că un client (sau inginerul responsabil de planificarea producției) plasează o comandă (pasul 1), după care solver-ul CSP calculează un plan de fabricație optimizat prin utilizarea datelor de configurare şi de mentenanță stocate în interiorul serverelor UA şi obținute prin intermediul serviciilor (paşii 2, 3 şi 4). După ce planul de fabricație este stabilit, un serviciu complex este lansat pentru a transfera soluția către serverul OPC UA corespunzător (şi implicit dispozitivelor, din nou tot prin intermediul serviciilor - **pasul 5**) şi pentru a monitoriza execuția comenzii (pasul 6). Dacă apar erori sau alarme, serviciul complex va efectua activități de gestionare a erorilor (pasul 7).

Încă de la început trebuie subliniat faptul că operațiile critice din punct al timpului de execuție sunt păstrate pe dispozitivele de control, arhitectura construită deasupra acestor dispozitive având rolul de a transmite comenzi de fabricație optimizate şi de a monitoriza funcționarea dispozitivelor.

Nivelul inferior este reprezentat de serverele OPC UA care modelează datele de la nivelul dispozitivelor şi astfel fiecare informație devine uşor accesibilă într-un mod unificat. Noua arhitectură OPC UA a fost introdusă ca un înlocuitor al specificațiilor existente bazate pe COM, fără a pierde din caracteristici sau din performanță [***The OPC Foundation: "OPC UA Specification: Part 1 – Concept", 2011. [Online]. Available at: http://opcfoundation.org/.]. OPC UA furnizează toate datele într-un spațiu de adrese unificat, spre deosebire de specificațiile OPC-ului clasic, care erau separate. În concluzie datele curente, alarmele & evenimentele şi datele istorice pot fi accesate în acelaşi mod şi

sunt corelate. Pe de altă parte OPC UA introduce un meta model extensibil în care fiecare element are un tip, precum în programarea orientată pe obiecte. Noua specificație UA este independentă de platformă şi de limbaj.

Datorită situațiilor diferite în care OPC este utilizat, noul standard este scalabil de la sistemele înglobate la SCADA şi DCS (Distributed Control Systems), la MES (Manufacturing Execution Systems) şi chiar ERP (Enterprise Resource Planning).

Unul dintre cele mai importante aspecte în ceea ce priveşte migrarea de la OPC-ul clasic la specificația UA a fost dezvoltarea unei strategii de migrare potrivite. În literatura de specialitate au fost menționate numeroase strategii [Mahnke, W., Leitner, S.H., Damm, M. *"OPC Unified Architecture"*, Springer Press, Berlin, Germany, 2009.]. Pentru arhitectura curentă s-a utilizat soluția *software special de adaptare*. Această soluție reprezintă o cale de mijloc între abordările *wrapper-proxy* (dezvoltare rapidă, capabilități de modelare reduse) şi abordarea *dezvoltare nativă* (timp mare de dezvoltare, capabilități estinse de modelare) deoarece poate fi dezvoltată rapid şi furnizează capabilități extinse de modelare. Singurul dezavantaj este că trebuie utilizat un server clasic COM, aspect care conduce la creşterea timpului de comunicație, dar testele de performanță care au fost efectuate demonstrează că acest lucru nu afectează performanța arhitecturii. Server-ele UA înglobează un client clasic OPC dezvoltat pe baza SDK-ului open source JEasyOPC [***JEasyOPC, 2011. [Online]. Available at: http://jeasyopc.sourceforge.net.] pentru a permite comunicarea cu server-ele OPC clasice bazate pe COM. Pentru dezvoltarea serverelor UA s-a utilizat SDK-ul (Software Development Kit) JAVA UA dezvoltat de către Prosys. Serverele UA dezvoltate pentru această arhitectură profită de capabilitățile de modelare unificate, ceea ce înseamnă că înglobează specificațiile *alarme & evenimente* şi *date istorice* [***The OPC Foundation: "OPC UA Specification: Part 9 - Alarms and Conditions", 2011. [Online]. Available at: http://opcfoundation.org/.], [***The OPC Foundation: "OPC UA Specification: Part 11 - Historical Access", 2011. [Online]. Available at: http://opcfoundation.org/.]. În arhitectura propusă implementarea specificației alarme & evenimente este crucială în special pentru gestionarea erorilor.

Spațiul de adrese este cel mai important concept pentru specificația UA şi toate celelalte funcții bloc trebuie să fie realizate pe baza acestuia [***The OPC Foundation: "OPC UA Specification: Part 3 - Address Space Model", 2011. [Online]. Available at: http://opcfoundation.org/.]. Unitățile de bază ale spațiului de adrese sunt nodurile şi referințele care leagă nodurile. Fiecare nod are atribute care sunt determinate de tipul său, dar există şi un set de atribute comune tuturor nodurilor, precum nodeid (identificatorul nodului). Există un nod de bază, care este un tip de nod abstract şi nu poate fi

instanțiat. Toate celelalte noduri din spațiul de adrese sunt derivate de la nodul de bază: tip de referință, obiect, tip de obiect, variabilă, tip de variabilă, metodă, tip de date şi vedere. Prin intermediul referințelor, toate nodurile de spațiul de adrese sunt organizate într-un mod structurat şi sunt legate între ele. Serverele UA complet funcționale conțin mii de noduri cu scopul de a oferi clienților toate informațiile de care au nevoie şi de aceea este important să se implementeze o modalitate eficientă de generare a spațiului de adrese, care să faciliteze de asemenea o întreținere uşoară a acestuia.

O serie de algoritmi au fost dezvoltați şi folosiți pentru a genera în mod automat şi eficient spațiul de adrese al unui server UA. Ideile de bază ale algoritmilor sunt: divizarea nodurilor în grupuri şi adăugarea secvențială a grupurilor în spațiul de adrese, precum şi numirea unei referințe principale pentru fiecare nod țintă.

Serverele OPC UA comunică direct cu serviciile, care din punctul de vedere al serverului reprezintă clienți OPC UA, şi cu dispozitivele fie direct, în cazul în care protocolul de comunicație cu acestea este implementat în serverul UA, fie indirect prin intermediul unui server OPC clasic, caz în care se foloseşte soluția *software special de adaptare*.

În general pentru o aplicație se recomandă folosirea unui singur server OPC UA, deoarece pe de o parte caracteristicile acestuia permit gestionarea unui număr relativ mare de echipamente, iar pe de altă parte coordonarea procesului de fabricație este simplificată. În cazul unei uzine cu numeroase sisteme de fabricație, se recomandă folosirea mai multor servere OPC UA, nu doar datorită performanțelor sporite, dar şi datorită separării clare a instalațiilor.

Cel de-al doilea nivel al arhitecturii este reprezentat de servicii care sunt organizate pe două nivele (în general numărul nivelelor este nelimitat, dar cantitatea corectă de granularitate a serviciilor este crucială pentru succesul arhitecturii [Krammer, A., Heinrich, B., Henneberger, M., Lautenbacher, F. *"Granularity of services an economic analysis"*, Bussiness & Information Systems Engineering, Vol.3, 2011, pp. 345-358.]). Ideea de a dezvolta o arhitectură SOA în joncțiune cu un server OPC UA pare naturală deoarece specificația UA a fost dezvoltată pe baza elementelor cheie SOA [Virta, J., Seilonen, I., Tuomi, A., Koskinen, K. *"SOA-Based integration for batch process management with OPC UA and ISA-88/95"*, IEEE Conference on Emerging Technologies and Factory Automation ETFA, Bilbao, Spania, Sept. 2010, pp. 1-8.]. Spre deosebire de vechea specificație, UA defineşte 51 de servicii ca mijloc principal de comunicație şi abstractizare dintre clienții şi serverele UA. Prin introducerea serviciilor ca nivel intermediar, întreaga arhitectură devine mai flexibilă şi asigură timpi de reacție mai mici când sunt necesare diverse modificări.

Primul nivel de servicii este compus din servicii de bază care efectuează operații de citire, scriere, etc. şi care vor fi integrate în orice scenariu de utilizare a arhitecturii: *ScrieNodVariabila* (modifică valoarea unui nod variabilă), *CitesteNodVariabila* (citeşte valoarea unui nod variabilă), *ApeleazaMetoda* (apelează o metodă cu un set de argumente de intrare şi preia valori corespunzătoare unui set de argumente de ieşire), *CitesteStareAlarma* (citeşte starea curentă a unei alarme), *CitesteAlarmaEveniment* (determină alarmele şi evenimentele înregistrate în trecut într-un anumit interval de timp), *CitesteDateNeprocesate* (citeşte toate datele istorice ale unui nod variabilă dintr-un anumit interval de timp), *CitesteDateProcesate* (determină o valoare procesată – sumă, minim, maxim, medie, etc. – corespunzătoare unui set de valori înregistrate pentru un nod variabilă într-un anumit interval de timp), *CitesteDateDeLaMomentul* (citeşte valoarea unui nod variabilă de la un anumit moment data din trecut, calculându-se o valoare interpolată în cazul în care la momentul specific nu s-a realizat nici o înregistrare).

Toate serviciile de bază cât şi unele complexe se comportă ca şi clienți din punctul de vedere al unui server UA. Fiecare serviciu care simultan reprezintă un client UA trebuie să stabilească o conexiune cu serverul UA pentru a putea extrage datele necesare clientului serviciului. Setarea unei conexiuni cu server-ul UA implică mai mulți paşi şi de asemenea necesită un anumit timp: trebuie generată o *descriere a aplicației*, după care, pe baza acestei descrieri, se va genera o *identitate a aplicației*. În consecință a fost folosită o abordare bazată pe sesiuni pentru a reduce timpul de răspuns pentru clientul serviciului: fiecare serviciu ține o evidență a conexiunilor, ceea ce înseamnă că fiecare serviciu va conține un set de conexiuni pentru entitățile/utilizatorii care îl apelează. Această abordare este simplă şi poate fi uşor programată la nivel de serviciu. În plus pot fi stabilite conexiuni specializate, particularizate pentru fiecare tip de serviciu. Creşterea numărului de conexiuni nu reprezintă un dezavantaj deoarece serverele UA pot gestiona un număr mare de conexiuni fără a suferi pierderi de performanță.

Serviciile complexe pot fi împărțite în patru categorii: servicii care controlează şi monitorizează producția, servicii de gestionare a erorilor, servicii care citesc date de configurare pentru modelele CSP aflate le nivelul superior al arhitecturii şi servicii care furnizează informații generale despre procesul de fabricație.

De obicei aceste servicii complexe interacționează cu mai multe servicii de bază pentru a-şi finaliza operațiile. Serviciile complexe din prima categorie recepționează soluțiile de la modelele CSP şi controlează şi monitorizează procesele de fabricație (acestea sunt în general servicii cu durată de execuție mai lungă).

Înainte de determinarea soluțiilor optimizate, modelul CSP apelează unul sau mai multe servicii pentru a determina starea curentă a dispozitivelor fizice şi a parametriza astfel modelul. De asemenea, serviciile pot fi folosite pentru a determina valorile indicatorilor cheie de performanță de performanță (KPI) la nivelul întreprindere. O problemă specifică referitoare la primul grup de servicii a fost monitorizarea alarmelor şi evenimentelor. Deşi aceste servicii vor apela mai multe servicii de bază pentru a realiza operațiile, serviciile de bază nu pot fi folosite pentru a detecta stările de urgență şi alarmele.

Astfel, când un serviciu complex este apelat, inițial acesta preia planul de fabricare optimizat determinat la nivelul superior, stabileşte o conexiune cu serverul UA şi se abonează la alarmele şi evenimentele care corespund acțiunii curente efectuate de serviciul complex. Apoi, înscrie soluția optimă în spațiul de adrese al server-ului OPC UA prin apelarea succesivă a serviciilor de bază de scriere şi porneşte procesul de fabricare. În cazul în care apare o eroare în timpul procesului de fabricație, serviciul complex fie opreşte execuția procesului şi notifică inginerul în legătură cu evenimentul respectiv, fie apelează un serviciu special care gestionează eroarea [Sasa, A., Juric, M.B., Krisper, M. *"Service-oriented framework for human task support and automation"*, IEEE Trans. on Industrial Informatics, Vol. 4, 2008, pp. 292-302.].

Se poate observa faptul că, după cum este prezentat în figura 2, serviciile furnizează date atât nivelului superior (CSP) cât şi nivelului inferior (UA), îndeplinind astfel practic rolul de comunicare între nivelul CSP (care determină soluții optime de fabricație) şi nivelul UA (care foloseşte soluțiile optime de fabricație).

Serviciile au fost implementate folosind două framework-uri de servicii web: Apache CXF (servicii bazate pe SOAP – Simple Object Access Protocol) şi Jersey (servicii web bazate pe REST – Representational State TRansfer) şi un framework de servicii Java denumit Apache River (cunoscut si sub numele de Jini). Pentru determinarea framework-ului optim în cazul arhitecturii propuse, s-au dezvoltat o serie de teste de performanță, în urma cărora s-a ales framework-ul Apache River deoarece conduce la timpi de execuție mai mici.

Nivelul superior al arhitecturii propuse este reprezentat de un set de modele CSP precum şi de un set de solver-e care sunt folosite pentru rezolvarea acestor modele. Programarea bazată pe constrângeri este o paradigmă utilizată la rezolvarea problemelor combinatorice de optimizare iar o instanță a unei probleme CSP, numită de obicei model, este descrisă de un set de variabile, un set de valori posibile pentru fiecare variabilă şi un set de constrângeri între variabile.

Pentru rezolvarea modelelor au fost evaluate solver-ele open source Choco şi Jacop precum şi biblioteca JLPI (Java Linear Programming Interface). Primele două solvere sunt generale şi pot rezolva atât modele liniare cât şi neliniare în timp ce biblioteca JLPI este optimizată pentru modele liniare, pentru care conduce la timpi de execuţie mai mici, dar nu poate fi folosită pentru modele neliniare. O dificultate importantă în găsirea unei soluţii optime pentru un anumit model CSP este faptul că timpul de execuţie creşte exponenţial cu creşterea complexităţii modelului [Brevet US7260562: Czajkowski, G.M., Miller, M.F. *"Solutions for constraint satisfaction problems requiring multiple constraints"*, 2007.].

În continuare sunt prezentaţi o serie de algoritmi utilizaţi pentru generarea spaţiului de adrese al serverului OPC UA. Problema pe care o rezolvă aceşti algoritmi este aceea de generare automată şi eficientă a spaţiului adrese, simplificând astfel atât definirea cât şi modificarea acestuia. Server-ele OPC UA complet funcţionale conţin mii de noduri pentru a putea furniza clienţilor toate informaţiile de care au nevoie. Acesta este motivul pentru care a fost dezvoltat un set de algoritmi care să permită generarea automată, şi deci eficientă, a spaţiului de adrese al unui server OPC UA. Pentru algoritmii realizaţi se poate folosi orice tip de sistem de stocare a datelor (în cazul de faţă s-a ales un sistem relaţional de gestionare de baze de date).

În continuare este descrisă procedura de pornire a unui server UA astfel încât să poată fi identificaţi paşii specifici care sunt rezolvaţi prin algoritmi. **Figura 3** prezintă în partea stângă principalele etape care trebuie parcurse iar în partea dreapta sunt prezentate acţiunile detaliate ale etapei de generare a spaţiului de adrese.

Primul pas care trebuie parcurs este acela de a genera un validator de certificate. Acesta este folosit de către server pentru a verifica certificatele clienţilor care doresc să se conecteze la server. După aceea serverul trebuie să genereze un certificat care va fi folosit de clienţi pentru validarea serverului. Acesta este realizat în paşii doi şi trei. Mai întâi este generată o descriere a aplicaţiei şi apoi o identitate a aplicaţiei (certificatul). Aceşti primi trei paşi fac parte din noile standarde de securitate introduse în specificaţia UA.

Apoi trebuie setate proprietăţile serverului, de exemplu: portul, numele serverului, adresele IP prin intermediul cărora acesta poate fi accesat precum şi serverul de identificare (pasul 4). După aceea, în cursul **pasului 5**, serverul este iniţializat şi spaţiul de adrese este generat.

Primele două activităţi, 5.1 si 5.2, afişate în dreptunghiul din partea dreaptă reprezintă paşi preliminari. În timpul tuturor celorlalte activităţi sunt adăugate noduri şi/sau referinţe în cadrul spaţiul de adrese.

Este important de observat faptul că toate nodurile generate în timpul activității de creare a spațiului de adrese sunt stocate in Map-uri (mai precis *HashMap*-uri) ca perechi cheie-valoare. Cheia fiecărui obiect este o variabilă de tip şir de caractere compusă astfel: *namespaceindex_NodeId* (adică indexul spațiului de nume urmat de NodeId-ul nodului). NodeId-ul fiecarui nod este unic în spațiul său de nume şi ca rezultat combinația index al spațiului de nume-NodeId este unică în spațiul de adrese. Există două motive pentru care este important să se stocheze toate nodurile în interiorul acestor Map-uri: nodurile sunt necesare în timpul generării spațiului de adrese (de ex. pentru a preciza tipul şi nodul părinte) şi anumite atribute trebuie să fie accesate şi schimbate în timpul funcționării ulterioare a serverului, fie din cauza acțiunilor efectuate de către clienți sau doar pentru că valorile dispozitivelor modelate în spațiu de adrese sunt schimbate (de ex. atributul *valoare* al nodurilor variabilă).

Este important de reținut faptul că nodurile care au clase de noduri diferite trebuie să fie adăugate într-o ordine specifică în spațiul de adrese şi chiar şi nodurile care au aceeaşi clasă de noduri nu pot fi adăugate în spațiul de adrese într-o ordine aleatoare.

Obiectele *NodeManager* trebuie create înainte de a începe definirea nodurilor de diferite clase de noduri (pasul 5.3). Există un *NodeManager* care reprezintă rădăcina ierarhiei, care conține toate nodurile de bază definite de specificația UA. Pe lângă acesta utilizatorul poate defini oricâți *NodeManager*-i doreşte, fiecare dintre ei având un spațiu de nume diferit. Fiecare nod trebuie să aparțină unui *NodeManager*, justificându-se astfel crearea sa la începutul procedurii. De asemenea, fiecare nod şi fiecare referință are câte un tip. Din această cauză ierarhiile de tipuri (modelele de informații) sunt definite înainte de a crea nodurile propriu-zise. Deoarece tipurile de noduri sunt de asemenea conectate prin intermediul referințelor, se vor crea mai întâi *ReferenceTypes* (pasul 5.4). Apoi pe parcursul paşilor 5.5 şi 5.6 sunt definite *ObjectTypes* (tipuri de obiecte) şi *VariablesTypes* (tipuri de variabile), atât cele simple cât şi cele complexe. În specificația UA nu există *MethodTypes* (tipuri de metode). Trebuie subliniat faptul că declarațiile de instanță sunt adăugate în cursul următoarelor activități, deci după ce întreaga ierarhie a tipurilor a fost creată. Fiecare declarație de instanță a unui supertip este valabilă şi pentru subtip. În acest caz fundația OPC a ales o abordare similară celei a limbajelor de programare orientate pe obiecte, şi anume variabilele supertipurilor sunt moştenite şi astfel nu este nevoie să se copieze codul în subtip. Prin urmare, declarațiile de instanță nu sunt duplicate în subtip (acest lucru se întâmplă doar dacă ele trebuie suprascrise). Pentru a obține o imagine clară a subtipului, clienții trebuie să solicite declarațiile de instanță ale supertipurilor. Acesta a fost aspectul care a permis generarea declarațiilor de instanță după ce a fost generată întreaga ierarhie de tipuri.

În continuare sunt create nodurile obiect şi nodurile variabilă (paşii 5.7 şi 5.8). În cursul fiecărei activităţi declaraţiile de instanţă sunt generate înaintea nodurilor obişnuite ale clasei respective de noduri. În final sunt generate nodurile metodă şi nodurile vedere (paşii 5.9 şi 5.10) şi de asemenea sunt modelate în spaţiul de adrese şi referinţele suplimentare (pasul 5.11).

Revenind în partea stângă a figurii 3, se observă că au mai rămas doi paşi de parcurs. În cursul pasului 6, sunt dezvoltate implementările metodelor (se va defini un *CallListener* adăugat la fiecare *NodeManager*: acesta conţine implementările tuturor nodurilor metodă definite în *NodeManagerul* respectiv). În final serverul este pornit şi clienţii se pot conecta la acesta (pasul 7).

În continuare se vor prezenta **activităţile preliminarii generării automate a spaţiului de adrese**. În partea dreaptă a figurii 3 sunt afişate două activităţi preliminarii (în pasul de creare a spaţiului de adrese). După cum s-a precizat anterior, toate nodurile spaţiului de adrese vor fi stocate în obiecte HashMap. Ca rezultat, prima activitate este reprezintată de crearea HashMap-urilor, câte unul pentru fiecare categorie de nod UA, având o cheie formată din obiecte String, iar ca valoare se folosesc diferite tipuri de noduri UA. Cea de-a doua activitate este de a popula aceste map-uri cu nodurile din modelul de informaţii de bază, adică din versiunea de bază a spaţiului de adrese, noduri care aparţin *NodeManager*-ului rădăcină. Această activitate este foarte importantă deoarece utilizatorii vor defini noduri care vor fi conectate la nodurile din modelul de informaţii de bază şi vor folosi referinţe care de asemenea aparţin aceluiaşi model de informaţii. Dacă acest pas nu este realizat corect, unele dintre nodurile definite de utilizator nu vor fi adăugate corect în spaţiul de adrese.

Suplimentar se desfăşoară o a treia activitate preliminară, care nu a fost menţionată în figura 3 deoarece nu aparţine procedurii de generare a spaţiului de adrese, dar care joaca un rol important în cazul algoritmilor dezvoltaţi. Aceasta se referă la stabilirea unei conexiuni cu sursa de date.

Tipurile de noduri reprezintă baza modelelor de informaţii incluse în serverul UA. În continuare vor fi descrise cele mai importante trei tipuri de noduri şi anume: *ObjectType*, *VariableType* şi *ReferenceType* (figura 4 prezintă convenţiile de modelare general acceptate [Mahnke, W., Leitner, S.H., Damm, M. *"OPC Unified Architecture"*, Springer Press, Berlin, Germany, 2009.]). *ReferenceTypes* sunt folosite pentru a reprezenta semantica referinţelor. Acestea sunt împărţite în două grupuri şi anume referinţe ierarhice şi referinţe non-ierarhice. Referinţele ierarhice sunt în principal folosite pentru a organiza spaţiul de adrese şi reflectă clasificarea spaţială şi logică a dispozitivelor. Spre deosebire de ierarhiile *ObjectType* şi *VariableType*, există deja o ierarhie de tipuri în ceea ce priveşte *ReferenceTypes* dar utilizatorul poate decide oricând faptul că are nevoie de noi referinţe, cu noi semantici.

În continuare va fi descris **algoritmul folosit pentru a genera ierarhia tipurilor de referințe**. Deoarece acest algoritm este foarte asemănător cu cele pentru generarea tipurilor de obiecte şi a tipurilor de variabile, acestea din urmă nu vor fi prezentate.

Se subliniază faptul că tipurile de referințe şi în general toate nodurile nu pot fi adăugate într-o ordine aleatoare în spațiul de adrese, deoarece în acest caz ar putea apărea situații în care ele trebuie conectate la noduri care nu sunt încă definite. În plus, nu se poate presupune faptul că nodurile sunt introduse în ordinea corectă în baza de date şi că această ordine asigură o generare corectă a spațiului de adrese. Prin urmare, este responsabilitatea algoritmului de a asigura că nodurile sunt adăugate în ordinea corectă în spațiul de adrese. Astfel, tipurile de referințe conectate la tipurile de referințe de bază sunt citite din baza de date (în fiecare algoritm funcțiile care încep cu sintagma *citeste* realizează interogări asupra sursei de date: `citesteRTConLaRTBaza()`). Apoi este deschisă o buclă, în interiorul căreia fiecare iterație gestionează un tip de referință. Se determină *NodeManager*-ul tipului de referință curent `RT[i]` (`obtineNodeManager(RT[i].nm`, literele care urmează după punct reprezintă mereu o prescurtare a proprietății (ex: `nm` reprezintă nodemanager), nodul părinte (`obtineNodParinte()`) şi referința prin intermediul căreia acest nod va fi conectat la nodul părinte (`obtineReferinta()`). Apoi sunt setate mai multe proprietăți precum *browsename* - `RT[i].bn`, *issymmetric* - `RT[i].sy`, *isabstract* - `RT[i].ab`, etc. În cele din urmă tipul de referință este adăugat în spațiul de adrese (`adaugaRTLaSA()`) şi la map-ul de tipuri de referințe. Acțiunea finală este de a adăuga tipul de referință la un grup de referințe care vor fi folosite în pasul următor al algoritmului (`adaugaRTLaGrup()`). După terminarea primei bucle *for*, au fost adăugate în spațiul de adrese toate tipurile de referințe conectate la cele de bază. În plus, grupul de referințe alcătuit în timpul buclei for este folosit pentru a interoga baza de date şi pentru a determina tipurile de referințe conectate la tipurile de referințe conținute în grup (`citesteRTConLaRTBaza()`). Astfel se garantează faptul că aceste noduri se pot adăuga spațiului de adrese deoarece nodurile lor părinte au fost deja create. După ce au fost citite aceste tipuri de referințe, grupul, care a fost stabilit anterior, este resetat pentru a pregăti generarea unui nou grup (`reseteazaGrupRT()`). Apoi aceste noduri sunt adăugate spațiului de adrese şi un nou grup este stabilit, grup care va fi folosit pentru următoarea interogare. În acest fel se continuă adăugarea grupurilor la ierarhia tipurilor de referințe până când nu mai există noduri de extras din baza de date. Acesta este motivul pentru care după prima buclă *for* se utilizează o bucla *while* cu un număr nedeterminat de iterații. Fiecare iterație a buclei *while* conține o bucla for similară cu cea folosită anterior buclei *while*.

În consecință, prin intermediul acestui algoritm ierarhia de referințe va fi generată corect, indiferent de cât de complicată este sau indiferent de cât de multe nivele conține (bucla *while* execută în general zeci de iterații până când se termină execuția acesteia).

Prima buclă *for* nu a fost inclusă în buclele *for* executate în interiorul buclei *while* deoarece tipurile de referințe adăugate prin intermediul primei bucle for sunt legate de tipurile de referință de bază, care nu sunt reprezentate ca rânduri în tabelul corespunzător din baza de date.

Prin urmare ele trebuie tratate diferit, deşi aparent funcțiile scrise în pseudo-cod sunt aceleaşi.

Algoritmul 1. Generarea tipurilor de referințe

```
citesteRTConLaRTBaza()
FOR i=0 la RTLaRTBaza-1
  obtineNodeManager(RT[i].nm)
  obtineNodParinte(RT[i].nm)
  obtineReferinta(RT[i].r)
  seteazaProprietatiRT(RT[i].bn, RT[i].sy, RT[i].ab,...)
  adaugaRTLaSA()
  adaugaRTLaGrup(RT[i])
END
WHILE GrupRT.dimensiune()>0
  citesteRTConLaRTBaza()
  reseteazaGrupRT()
  FOR i=0 la RT_In_Grup-1
    obtineNodeManager(RT[i].nm)
    obtineNodParinte(RT[i].nm)
    obtineReferinta(RT[i].r)
    seteazaProprietatiRT(RT[i].bn, RT[i].sy, RT[i].ab,...)
    adaugaRTLaSA()
    adaugaRTLaGrup(RT[i])
  END
END
```

Pe lângă tipurile de referință, de obiect şi de variabilă, şi obiectele şi variabilele sunt generate cu algoritmi similari. Principala diferență în cazul acestor ultime două categorii de noduri este ca acestea pot fi noduri declarații de instanță (daca sunt conectate direct sau indirect cu un tip de obiect, respectiv variabila) sau noduri obişnuite. Cele două cazuri sunt tratate separat dar prin algoritmi similari.

Nodurile metodă (figura 5) reprezintă metode, adică entități care pot fi apelate de clienți și care returnează un rezultat. Pentru fiecare metodă trebuie specificate argumentele de intrare și de ieșire. Metodele au fost introduse în ideea că reprezintă elemente de cod care se execută foarte rapid. În mod normal ele fac parte din structura unui obiect complex sau a unui tip de obiect, ceea ce înseamnă că și metodele pot fi declarații de instanță.

Algoritmul 2 ilustrează pașii necesari pentru **generarea nodurilor metodă**. Astfel, nodurile care reprezintă declarații de instanță și metodele comune trebuie din nou tratate separat. Deoarece metodele nu pot avea ca nod părinte alte metode, nu mai este nevoie să se aplice o strategie similară celei utilizate pentru primii doi algoritmi, adică să se adauge noduri în grupuri succesive. Prin urmare, declarațiile de instanță (`IDM[i]`) sunt citite din baza de date (`citesteMetodeID()`) și după ce sunt setate proprietățile comune tuturor nodurilor (nodemanager, nod părinte, referință principală, proprietățile nodului, etc.), sunt specificate argumentele de intrare și de ieșire. Deoarece, în general, o metodă poate avea mai multe argumente de intrare și de ieșire, acestea sunt stocate în tabele separate. La început sunt citite argumentele de intrare (`readInputArg()`), apoi sunt setate proprietățile fiecărui argument (`seteazaProprietatiArg()`: nume, tip de dată, descriere, etc.) și în final se setează tabloul de argumente de intrare (`IDM[i].seteazaArgIntrare(ArgIntrare)`). După ce sunt efectuate aceleași operații și pentru argumentele de ieșire (`ArgIesire[j]`), sunt specificate regulile de modelare referite de către metodă, la fel ca și pentru declarațiile de instanță de tip variabilă sau obiect: mai întâi acestea sunt citite de la sursa de date (`citesteReguliModelare()`) și apoi se adaugă referințe între nodul metodă și regulile de modelare (`referireRegulaModelare()`). În final, declarațiile de instanță sunt adăugate la spațiul de adrese (`adaugaMLaSA()`).

Apoi sunt adăugate la spațiul de adrese și metodele comune folosind un algoritm asemănător, singura diferență fiind faptul că nu au fost precizate reguli de modelare.

Algoritmul 2 – Generarea metodelor

```
citesteMetodeID()
FOR i=0 la MetodeID-1
  ...
  citesteArgIntrare(IDM[i])
  FOR j=0 la ArgIntrare-1
      seteazaProprietatiArg(ArgIntrare[j].nume,...)
  END
  IDM[i].seteazaArgIntrare(ArgIntrare)
```

```
  citesteArgIesire(IDM[i])
  FOR j=0 la ArgIesire-1
      setArgProperties(ArgIesire[j].nume,...)
  END
  IDM[i].seteazaArgIesire(ArgIesire)
  citesteReguliModelare(IDM[i])
  FOR j=0 la RM-1
    referireRegulaModelare(IDM[i]. MR[j])
  END
  adaugaMLaSA()
END
citesteMetode()
FOR i=0 la Metode-1
  ...
END
```

Folosind aceşti algoritmi, spaţiul de adrese poate fi modificat atât online (adică în timpul funcţionării serverului), caz în care la oprirea server-ului noua structură a spaţiului de adrese este salvată în sursa de date, cât şi offline, prin modificarea directă a conţinutului sursei de date.

Invenţia prezintă următoarele avantaje:

- Dispozitivele de control existente în cadrul întreprinderilor pot fi integrate în interiorul arhitecturii folosind soluţia *software special de adaptare* şi un server OPC clasic intermediar plasat din punct de vedere conceptual între dispozitive şi serverul OPC UA;
- Spaţiul de adrese care modelează datele de la nivelul dispozitivelor este generat în mod eficient folosind un set de algoritmi dezvoltaţi special în acest scop;
- Deoarece operaţiile critice din punctul de vedere al timpului de execuţie sunt păstrate pe dispozitivele de control, se elimină problemele întârzierilor nedorite introduse de serviciile software (în special de serviciile web). Avantajele oferite de utilizarea serviciilor software sunt însă păstrate prin introducerea acestora la nivelul intermediar al arhitecturii;
- Planurile optime de fabricaţie, care sunt determinate prin modele CSP sau prin orice altă tehnică, pot fi utilizate automat (fără intervenţia unui operator uman) prin intermediul arhitecturii propuse;

- Arhitectura permite monitorizarea simplă a indicatorilor cheie de performanță, care trebuie raportați la nivelul de management al întreprinderii. Acest avantaj este asigurat în principal de nivelul OPC UA, unde se centralizează datele, şi de nivelul serviciilor care permite extragerea informațiilor;
- Arhitectura este flexibilă, adaptabilă şi reutilizabilă, aspecte care permit reducerea timpilor de instalare şi setare la valori minime. După cum este prezentat în figura 1, toate cele trei nivele (tehnologii) contribuie la acest avantaj.

Se dă în continuare un exemplu de realizare a invenției prin intermediul unei aplicații industriale. Aplicația a fost formulată pe baza liniei flexibile FMS-200. Pentru a testa arhitectura în contextul unei aplicații complexe, s-a formulat o problemă care utilizează mai multe linii flexibile, una dintre ele fiind linia FMS 200. Astfel se presupune că o uzină dispune de patru linii flexibile care pot efectua în total cinci operații diferite pentru fabricarea a trei tipuri diferite de mecanisme de rotație (figura 6 prezintă un astfel de mecanism de rotație). Primele două linii flexibile pot efectua primele două operații: frezare şi găurire. Celelalte două linii flexibile efectuează cele trei operații rămase: plasarea şuruburilor, fixarea şuruburilor şi inspecția produsului final. Diferența principală dintre cele trei tipuri de produse este dată de numărul de şuruburi care trebuie montate şi înşurubate (două, trei sau patru şuruburi).

Un client plasează o comandă prin care specifică numărul de piese dorite din fiecare tip. În aceste condiții se va formula o problemă al cărei scop este de a determina un orar de lucru care să permită fabricarea automată a pieselor, fără intervenția unui operator uman, într-un timp cât mai scurt.

Trei dintre cele patru linii flexibile vor fi simulate iar cea de-a patra linie este linia flexibilă FMS 200. Această linie conține trei celule de fabricație care efectuează ultimele trei operații necesare pentru finalizarea produsului. Figura 7 prezintă structura completă utilizată în cadrul aplicației. Un server OPC UA modelează în interiorul spațiului său de adrese toate datele corespunzătoare celor patru linii flexibile. Simulatoarele construite în interiorul serverului OPC UA reacționează la modificările valorilor conținute în spațiul de adrese, introducând timpi de întârziere egali cu suma dintre durata operațiilor şi timpii de comunicare cu dispozitivele de control. Astfel, din punctul de vedere al arhitecturii software pentru care serverul OPC UA reprezintă nivelul de bază, nu se face nici o distincție între liniile flexibile simulate şi cea reală. Pentru stabilirea comunicației dintre serverul OPC UA şi linia flexibilă FMS 200 se foloseşte soluția *software special de adaptare*, ceea ce înseamnă că se introduce un server OPC clasic între serverul OPC UA şi dispozitivele de control ale celulelor.

Dispozitivele de control ale celulelor liniei flexibile FMS 200 sunt automate programabile Siemens S7-300. Pe lângă automatele programabile ale celulelor există şi un master (tot un automat programabil S7-300) care coordonează acţiunile celulelor şi care controlează banda de transport utilizată pentru mutarea pieselor de la o celulă la alta. Comunicaţia între toate dispozitivele de control se realizează printr-o reţea Profibus. Celulele liniei flexibile pot fi folosite atât individual, cât şi în cadrul unui scenariu de fabricaţie complex, caz în care operaţiile celulelor sunt controlate de master.

Situaţia prezentată în figura 7 este una avantajoasă deoarece este suficient ca serverul OPC UA să comunice cu master-ul liniei flexibile. Acesta colectează toate datele relevante de la dispozitivele de control ale celulelor, astfel încât nu mai este nevoie să se stabilească separat conexiuni directe cu acestea. Comunicaţia dintre serverul OPC clasic şi master se realizează printr-un adaptor MPI-RS232 iar comunicaţia dintre serverul OPC clasic şi serverul OPC UA agregator se realizează prin intermediul unui client OPC clasic construit în cadrul serverului, care comunică cu serverul clasic cu ajutorul SDK-ului JEasyOPC.

Tabelul 1 prezintă timpii de execuţie ai celor cinci operaţii, pentru cele trei produse diferite şi pentru cele patru staţii (în continuare prin staţii se va face referire la liniile flexibile ale aplicaţiei).

Tab. 1. Timpii de execuţie ai operaţiilor pe cele patru staţii [s]

Staţie	Staţia A			Staţia B		
Produs	Mec. de Rotaţie 1	Mec. de Rotaţie 1	Mec. de Rotaţie 1	Mec. de Rotaţie 1	Mec. de Rotaţie 1	Mec. de Rotaţie 1
Frezare	80s	80s	80s	60	60s	60s
Găurire	15s	20s	25s	20	30s	40s
Staţie	Staţia C			Staţia D		
Plasare şuruburi	10s	15s	20s	15s	20s	25s
Robot	40s	50s	60s	30s	40s	50s
Inspecţie	10s	15s	20s	10s	15s	20s

Se poate observa faptul că aceşti timpi diferă în funcţie de operaţie, de staţie dar şi de piesa fabricată. În plus, pentru a modela corect procesul de fabricaţie trebuie incluşi şi timpi de transport, atât între celulele unei staţii (unde se foloseşte o bandă transportoare controlată de către master) cât şi între staţii. Timpii de transport sunt prezentaţi în tabelul 2. Aceşti timpi sunt independenţi de staţii, de aceea se prezintă doar timpii de transport între cele cinci operaţii.

Tab. 2. Timpi de transport între celulele stațiilor [s]

Operațiile între care se face transportul	Timp
Frezarea → Găurire	5
Găurire → Plasare şuruburi	20
Plasare şuruburi → Robot	4
Robot → Inspecție	5

În continuare va fi prezentat **modelul MIP** care poate fi utilizat pentru a determina un orar de lucru optim. Modelul generalizat a fost preluat din [Sawik, T. "*Scheduling in supply chains using mixed integer programming*", Wiley, Hoboken, NY, USA, 2011.] şi apoi a fost adaptat la problema curentă. Unul din cele mai importante aspecte care a necesitat modificări a fost introducerea timpilor de transport, atât între stații cât şi între celulele unei stații. Practic un sistem flexibil de fabricație flexibil (FAS) poate fi considerat ca fiind o problemă de tip job-shop flexibil. Orarul de executare a operațiilor pe maşinile existente trebuie să respecte următoarele constrângeri: o stație poate procesa cel mult o operație la un moment dat şi procesarea unei operații nu poate fi întreruptă de o altă operație. Funcția obiectiv are ca scop obținerea unui timp de final al producției care să fie minim.

Un FAS generalizat este o rețea de etape de asamblare care sunt interconectate prin elemente de transport, unde fiecare etapă constă dintr-un număr de maşini identice şi paralele. Fiecare stație/maşină are un spațiu de lucru finit pentru furnizorii de componente şi pentru zonele tampon de intrare şi de ieşire care trebuie să stocheze produsele care aşteaptă să fie procesate şi respectiv care trebuie să fie trimise către alte stații.

Trebuie subliniat faptul că un sistem FAS este un sistem de producție multidirecțional care permite revenirea la stațiile parcurse anterior. Un alt aspect important este acela că un sistem FAS poate asambla simultan şi la o rată ridicată, o varietate largă de tipuri de produse, ceea ce reprezintă un avantaj important în comparație cu liniile de transfer convenționale care sunt proiectate pentru volume mari de lucru dar cu varietate de fabricare scăzută.

Cei mai importanți termeni pentru un sistem FAS, din punct de vedere al determinării orarului de lucru, sunt *planificarea resurselor* şi *planificarea temporală*. Presupunând că trebuie asamblate diverse produse, obiectivul problemei de *planificare a reurselor* este acela de a asocia operații de asamblare şi furnizori de componente stațiilor de asamblare cu spații de lucru limitate, scopul fiind

acela de a determina rutele de asamblare pentru diverse produse astfel încât să se obțină un volum de lucru echilibrat între stații [Brevet US6467605: Claude, D. "*Process of manufacturing*", 2002.].

Obiectivul problemei de *planificare temporală* este acela de a determina secvența detaliată și momentele de timp pentru toate operațiile de asamblare ale fiecărui produs, astfel încât productivitatea sistemului să fie maximizată.

La implementarea modelului MIP a fost utilizată o abordare ierarhică în cadrul căreia sunt folosite două modele, mai întâi unul pentru echilibrarea volumul de lucru al stațiilor și apoi un model pentru determinarea orarului de lucru cu durata cea mai scurtă de realizare a operațiilor, folosindu-se asocierea dintre operații și stații determinată în cadrul primului model. În continuare vor fi prezentate cele două modele MIP:

- L1a: model ierarhic pentru planificare;
- S|L1: model ierarhic pentru scehduling.

Datele de intrare ale algoritmilor L1a, S|L1 sunt prezentate în cele ce urmează:

Indici:

- f= operație de asamblare $f \in F$;
- i= stație de asamblare, $i \in I = \{1,...,m\}$;
- k= produs, $k \in K = \{1,...,n\}$.

Parametrii de intrare:

- a_{if} = spațiul de lucru necesar efectuării operației f pe stația i;
- b_i = spațiul de lucru total al stației i;
- e_{ifk} = timpul de finalizare cel mai mic pe stația i a operației f ce aparține produsului k;
- q_{ifk} = timpul de asamblare pe stația i a operației f ce aparține produsului k;
- t_{fg} = timpul de transport dintre operația f și operația g;
- I_f = subsetul de stații capabile să efectueze operația f;
- F_k = subsetul de operații necesare produsului k;
- Q = o constantă pozitivă mai mare decât timpul total de producție;
- R_k = un set de perechi de operații predecesor-succesor (f, g) pentru produsul k unde f și g sunt operații succesive in interiorul subsetului F_k.

Variabile de decizie

- c_{ifk} = timpul de finalizare pe stația i a operației f necesare produsului k;
- x_{ifk} = variabilă de asociere a produselor, care este egală cu 1 dacă produsul k este asociat stației i pentru a efectua operația f; altfel $x_{ifk} = 0$;
- y_{fkgl} = variabilă de secvențiere a operațiilor, care este egală cu 1 dacă operația f a produsului k precede operația g a produsului l când ambele operații sunt asociate aceleiași stații; altfel $y_{fkgl} = 0$;
- z_{if} = variabilă de asociere a operațiilor, care este egală cu 1 dacă operația f este asociată stației $i \in I_f$; altfel $z_{if} = 0$.

Modelul L1a este utilizat pentru a realiza planificarea operațiilor. Se definește funcția obiectiv: minimizarea volumului de lucru maxim reprezentat prin variabila W. În continuare sunt definite constrângerile modelului.

Constrângeri de asociere a operațiilor

Aceste constrângeri asigură faptul că fiecare operație este asociată cel puțin unei stații de asamblare (aceeași operație poate fi efectuată pe stații diferite pentru produse diferite deoarece sunt admise rute alternative) și că spațiul de lucru total necesar efectuării operațiilor pe fiecare stație de asamblare nu poate depăși spațiul de lucru finit al fiecărei stații.

$$\sum_{i \in I_f} z_{if} \geq 1; \quad f \in F \tag{1}$$

$$\sum_{i \in I_f} a_{if} z_{if} \leq b_i; \quad i \in I \tag{2}$$

Constrângeri pentru asocierea produselor

Cel de-al doilea pas constă în definirea constrângerilor pentru asocierea produselor. Acestea asigură faptul că fiecare produs este asociat unei singure stații pentru a efectua o anumită operație și că fiecare produs poate fi direcționat către stațiile unde vor fi efectuate operațiile necesare finalizării sale:

$$\sum_{i \in I_f} x_{ifk} = 1; \quad k \in K, f \in F_k \tag{3}$$

$$x_{ifk} \leq z_{if}; \quad k \in K, f \in F_k, i \in I_f \tag{4}$$

În plus se adaugă un set de constrângeri care asigură că pentru stațiile identice din punctul de vedere al operațiilor care pot fi executate, operațiile unui anumit produs se efectuează mereu pe aceeași stație.

$$x_{ijk} = x_{i(j+1)k}; \quad k \in K, i \in I_f, j = 0,2,3 \tag{5}$$

Constrângeri referitoare la volumul de lucru maxim

Cel de-al treilea pas constă în specificarea constrângerilor care iau în calcul volumul de lucru maxim al fiecărei stații. Astfel, pentru fiecare stație, timpul total de asamblare necesar efectuării operațiilor, nu poate depăși volumul de lucru maxim (care urmează a fi minimizat).

$$\sum_{k \in K} \sum_{f \in I_k} q_{ifk} x_{ifk} \leq W; \quad i \in I \tag{6}$$

Constrângeri de pozitivitate şi integritate a variabilelor

$$x_{ifk} \in \{0,1\}; \quad k \in K, f \in F_k, i \in I_f \tag{7}$$

$$z_{if} \in \{0,1\}; \quad f \in F, i \in I_f \tag{8}$$

$$W \geq 0$$

Model S|L1 este utilizat pentru planificare temporală a operațiilor. Se definește funcția obiectiv: minimizarea variabilei c_{max}, care reprezintă cel mai mare timp de finalizare a unei operații. În continuare sunt definite constrângerile modelului.

Constrângeri pentru evitarea suprapunerii procesării pieselor

Aceste constrângeri asigură faptul că oricare două operații a două produse diferite asociate aceleiaşi stații nu pot fi prelucrate simultan:

$$c_{ifk} + Qy_{fkgl} \geq c_{igl} + q_{ifk}; \quad k,l \in K, f \in F_k, g \in F_l, i \in I : k < l, x^L_{ifk} x^L_{igl} = 1 \tag{9}$$

$$c_{igl} + Q(1 - y_{fkgl}) \geq c_{ifk} + q_{igl}; \quad k,l \in K, f \in F_k, g \in F_l, i \in I : k < l, x^L_{ifk} x^L_{igl} = 1 \tag{10}$$

Constrângerile sunt definite doar pentru acele perechi de operații (f, g) care sunt asociate aceleiaşi stații i (operația f aparține produsului k şi operația g ce aparține produsului l), adică în cazul în care $x^L_{ifk} x^L_{igl} = 1$.

Constrângeri pentru finalizarea produsului

Aceste constrângeri asigură faptul că timpul de finalizare al fiecărei operații a unui produs pe o anumită stație nu poate fi mai mic decât timpul cel mai recent de finalizare pe acea stație, că efectuarea fiecărei operații a fiecărui produs nu poate fi începută înainte de momentul de finalizare a procesării operației precedente, la care se adaugă şi timpul de transport, şi că operațiile succesive ale fiecărui produs, asociate aceleiaşi stații, sunt efectuate fără pauze.

$$c_{ifk} \geq e_{ifk}; \quad k \in K, f \in F_k, i \in I_f : x^L_{ifk} = 1 \tag{11}$$

$$c_{hgk} - q_{hgk} \geq c_{ifk} + t_{fg}; \quad k \in K, (f,g) \in R_k, i,h \in I : i \neq h, x^L_{jfk} x^L_{hgk} = 1 \tag{12}$$

$$c_{igk} - q_{igk} = c_{ifk} + t_{fg}; \quad k \in K, (f,g) \in R_k, i \in I : x^L_{jfk} x^L_{hgk} = 1 \tag{13}$$

Ultimele două constrângeri mențin relațiile de precedență dintre operațiile aceluiași produs.

Constrângeri referitoare la timpul maxim de finalizare a produselor

Cel de-al treilea pas al modelului constă în specificarea timpului maxim de finalizare a produselor. Astfel, durata de prelucrare este determinată de timpul maxim de finalizare al tuturor produselor şi durata de prelucrare nu poate fi mai mică decât volumul de lucru maxim.

$$c_{ifk} \leq c_{\max}; \quad k \in K, f \in F_k, i \in I_f : x^L_{ifk} = 1 \tag{14}$$

$$c_{\max} \geq W^L \tag{15}$$

unde W^L este valoarea soluției obținute ca urmare a rezolvării modelului L1a.

Constrângeri de pozitivitate şi integritate a variabilelor

$$c_{ifk} \geq 0; \quad k \in K, f \in F_k, i \in I_f : x^L_{ifk} = 1 \tag{16}$$

$$y_{fkgl} \in \{0,1\}; \quad k,l \in K, f \in F_k, g \in F_l : \sum_{i \in I} x^L_{ifk} x^L_{igl} = 1 \tag{17}$$

Timpul de finalizare cel mai mic e_{ifk} se calculează cu următoarea formulă şi reprezintă o constantă a modelului MIP:

$$e_{ifk} = q_{ifk} + \sum_{f' \in F_k : f' < f} \min_{i \in I_{f'}} (q_{if'k}) \tag{18}$$

Această expresie semnifică faptul că timpul de finalizare cel mai mic e_{ifk} este suma dintre timpul de asamblare pentru stația, produsul şi operația curentă şi timpul minim total de asamblare al tuturor operațiilor $f' \in F_k$ ale produsului curent, care preced operația curentă.

În continuare sunt prezentate o serie de **detalii de implementare.** În figura 8 este prezentată secvența de operații urmate pentru fabricarea pieselor comandate de un client. Inițial se apelează un serviciu complex care citeşte datele de configurare ale modelelor MIP din spațiul de adrese al serverului OPC UA (acest serviciu complex este format din apeluri succesive ale serviciului de bază de citire *CitesteNodVariabila*). Apoi se execută modelul MIP şi se determină un orar de lucru optim. În continuare se porneşte serviciul complex care preia soluția modelelor MIP şi o inserează în spațiul de

adrese al serverului prin apelul repetat al serviciului de bază de scriere (*ScrieNodVariabila* – apelarea se realizează din interiorul firului de execuție al solver-ului MIP). În final se pornește fabricarea pieselor, se monitorizează execuția, intervenindu-se în cazul apariției unei erori.

Spațiul de adrese al serverului OPC UA este unul dintre cele mai importante concepte ale întregii arhitecturi. După cum se poate observa în figura 7, a fost folosit un singur server UA pentru întreaga aplicație. În general alegerea unui singur sau a mai multor servere UA depinde de complexitatea aplicației și a dispozitivelor, aspecte reflectate și de complexitatea spațiului de adrese. În general se recomandă utilizarea unui singur server UA pentru o aplicație, dar aceasta nu este o abordate obligatorie.

Figura 9 ilustrează spațiul de adrese simplificat al aplicației curente. La pornirea serverului OPC UA, acesta este creat prin intermediul algoritmilor de generare a spațiului de adrese prezentați anterior. Spațiul de adrese este divizat în trei părți principale:

- *Operații*: date privind cele cinci operații care trebuie efectuate: spațiul de lucru, timpul de procesare al fiecărei operații și timpul de transport. Deoarece spațiul de lucru ocupat de o operație depinde de stația pe care se va executa acea operație, această variabilă reprezintă de fapt un tablou unidimensional de variabile, a cărui dimensiune este dată de numărul stațiilor (patru în cazul de față). Deoarece sunt cunoscute stațiile pe care se pot efectua operațiile, spațiul de lucru al unei operații pentru o anumită operație va fi egal cu 1 dacă operația se poate executa pe acea stație și 1000 în caz contrar (1000 este considerată o valoare întreagă foarte mare și este folosită în loc de infinit). De exemplu operația de frezare va avea spațiul de lucru egal cu 1 pentru stațiile A și B și egal cu 1000 pentru stațiile C și D. Pe baza informațiilor conținute în aceste variabile se va inițializa parametrul de intrare a_{if} al modelelor MIP. În ceea ce privește timpul de procesare, acesta depinde atât de produs cât și de stație. În aceste condiții variabilele *Timp procesare* de sub fiecare operație sunt de fapt tablouri bidimensionale, care conțin timpul de procesare al operației respective pe fiecare stație și pentru fiecare produs. În cazul în care o operație nu se poate efectua pe o anumită operație (de exemplu frezarea nu se poate efectua pe stațiile C și D), atunci se va introduce o valoare foarte mare (1000) în locația respectivă. Pe baza informațiilor conținute în aceste variabile se va inițializa parametrul de intrare q_{ifk}. În ceea ce privește timpul de transport se folosește o variabilă care reprezintă de fapt un tablou bidimensional și care stochează timpii de transport între oricare două operații succesive. Pe baza informațiilor conținute în aceste variabile se va inițializa parametrul de intrare t_{fg}. În cazul în care nu se poate

efectua transportul între anumite operații, se va introduce valoarea 1000 în locația corespunzătoare.

- *Produse*: date privind operațiile necesare fiecărui produs. Variabilele *Op. 1 – Op. 5* sunt variabile booleene ale căror valori indică dacă o anumită operație este necesară la fabricarea unui anumit produs. În cazul de față toate operațiile sunt necesare pentru fiecare produs, însă folosind această abordare, spațiul de adrese poate fi folosit și pentru alte procese tehnologice, unde nu trebuie executate toate operațiile pentru fiecare produs.
- *Stații*: date privind operațiile efectuate pe fiecare stație și spațiul de lucru total disponibil pentru acea stație. Pe baza spațiului de lucru total, se va inițializa parametrul de intrare b_i al modelelor MIP. Deoarece pentru aplicația curentă spațiul de lucru ocupat de fiecare operație este egal cu 1 dacă operația se poate efectua pe stația respectivă, spațiul total de lucru al unei stații va fi egal cu numărul total de operații care se pot efectua pe acea stație (2 pentru stațiile A și B și 3 pentru stațiile C și D). În plus, pentru fiecare stație există o serie de obiecte care reprezintă celulele de fabricație ale stației respective.

Pe baza informațiilor extrase din spațiul de adrese se pot astfel inițializa toți parametri de intrare ai modelelor MIP utilizate pentru această aplicație. Inițializarea parametrilor a_{if}, b_i, q_{ifk} și t_{fg} a fost deja descrisă, parametrul e_{ifk} este calculat cu ajutorul ecuației (18) iar parametrii I_f, F_k și R_k sunt calculați pe baza parametrilor de intrare determinați din spațiul de adrese.

Variabila *Componente procesate* a fost inclusă ca exemplu pentru o variabilă KPI: aceasta numără câte piese au fost procesate pe stația respectivă. Această valoare poate fi apoi citită la sfârșitul lunii (sau după un anumit interval de timp) de către un serviciu pentru a fi raportată la nivelul ERP al întreprinderii.

Procesul de fabricare a mecanismelor de rotație comandate de client este supervizat de un **serviciu complex** dezvoltat special pentru această aplicație și prezentat în figura 10. Inițial serviciul se conectează la serverul UA și se abonează la variabila *Final* precum și la alarmele tuturor stațiilor și celulelor utilizate în aplicație. Apoi rezultatele MIP preluate ca parametru de intrare (variabila c_{ifk} reprezintă momentele de final ale execuției operațiilor, dar fiind date duratele de procesare, se pot determina și momentele de start) sunt înscrise în nodurile variabilă corespunzătoare celor trei stații (nodurile *Rezultate MIP* ale celulelor). În continuare este setată variabila *Start* și prin urmare procesul de fabricație începe. În timpul procesării, serviciul este capabil să intervină în cazul în care apare o alarmă prin apelarea unui serviciu special. Dacă nu este generată nici o alarmă atunci fabricarea pieselor se termină normal, după cum a fost programată și ca urmare variabila *Final* devine 1 și

serviciul complex îşi termină operaţia (variabila *Final* devine 1 atunci când nodurile variabilă *Ciclu nefinalizat* ale celulelor de inspecţie ale staţiilor C şi D devin 0 după inspecţia ultimei piesei conform planului de fabricare conţinut în variabilele *Rezultate MIP*).

Acest serviciu complex arată că o conexiune UA între un serviciu complex şi serverul UA nu este necesară doar pentru gestionarea erorilor ci şi pentru a aştepta o notificare din partea procesului, referitoare la finalizarea fabricării.

În figura 11 este prezentată interfaţa grafică utilizată pentru plasarea unei comenzi de către client sau de către inginerul care gestionează procesul de fabricare. În partea stângă a ferestrei se introduce cantitatea dorită de piese din fiecare tip după care se apasă butonul *Start*. În continuare se citesc datele de configurare din spaţiul de adrese şi se rezolvă modelul MIP. Ca urmare se afişează timpul total de fabricare precum şi timpii de fabricare ai fiecărei staţii şi se porneşte procesul de fabricare prin apelul serviciului complex corespunzător. Interfaţa grafică conţine şi o căsuţa text unde se afişează operaţia curentă aflată în execuţie (citire date de configurare, rezolvare model MIP, fabricare piese, fabricare finalizată), precum şi o căsuţă text unde se afişează evenimentele sau alarmele care apar de-a lungul fabricării.

În continuare sunt prezentate **rezultatele modelelor MIP** descrise. În acest sens modelele MIP prezentate au fost rezolvate iniţial cu ajutorul a trei solver-e şi rezultatele sunt prezentate în tabelul 3 pentru diferite numere de piese comandate (întotdeauna a fost specificat un număr egal de piese din fiecare tip).

Tab. 3. Timpii de execuţie ai solver-elor [s]

Solver / Piese	Choco	JaCoP	JLPI
9	0.88	0.85	0.21
18	13.74	13.12	5.08
36	59.23	57.63	17.67
72	421.4	412.9	91.34

Timpii afişaţi în tabel pentru abordarea ierarhică reprezintă suma timpilor de execuţie ai celor două modele (L1a şi S|L1), soluţionarea problemei de planificare temporală (S|L1) reprezentând întotdeauna aproximativ 80% din timpul total. Rezultatele din tabel arată că solver-ul JLPI este cel mai rapid deoarece acesta este un solver special destinat modelelor liniare. Celelalte două solvere sunt apropiate ca şi performanţă, cu uşoare avantaje pentru solver-ul JaCoP. Totuşi dacă solver-ul JLPI nu poate fi utilizat (dacă modelul este neliniar), solver-ul Choco este de preferat deoarece acesta dispune de mai

multe constrângeri, modelul este mai uşor de construit şi se economiseşte timp important la implementarea problemei.

În tabelul 4 sunt prezentaţi timpii de execuţie ai componentelor principale ale aplicaţiei. Rezultatele arată că peste 97.5% din timpul total este reprezentat de fabricarea efectivă a pieselor.

După cum este prezentat în tabel, acest procentaj se schimbă odată cu modificarea numărului de piese fabricate, în general fiind mai mic pentru un număr mai mare de piese deoarece timpul de execuţie al solver-ului creşte exponenţial iar timpul de fabricare creşte liniar. În cazul comenzilor foarte mari, pentru a evita scăderea semnificativă a procentajului reprezentat de timpul destinat fabricării efective a pieselor, se recomandă împărţirea comenzii în mai multe şarje (de câte 50 sau 100 de piese). În acest fel se poate garanta că întotdeauna majoritatea timpului este destinată fabricării pieselor.

Tab. 4. Timpii de execuţie ai componentelor principale ale aplicaţiei

Piese	**Operaţie**	**Timp [s]**	**Procentaj [%]**
9	Citirea parametrilor de intrare	3.23	0.587
	Rezolvarea modelelor MIP	0.21	0.038
	Scrierea soluţiei MIP în spaţiul de adrese	2.98	0.541
	Fabricare piese	544.0	98.83
18	Citirea parametrilor de intrare	3.23	0.301
	Rezolvarea modelelor MIP	5.08	0.474
	Scrierea soluţiei MIP în spaţiul de adrese	3.45	0.322
	Fabricare piese	1060.0	98.90
36	Citirea parametrilor de intrare	3.23	0.151
	Rezolvarea modelelor MIP	17.67	0.828
	Scrierea soluţiei MIP în spaţiul de adrese	4.23	0.198
	Fabricare piese	2108.0	98.82
72	Citirea parametrilor de intrare	3.23	0.075
	Rezolvarea modelelor MIP	91.34	2.119
	Scrierea soluţiei MIP în spaţiul de adrese	5.47	0.127
	Fabricare piese	4210.0	97.678

În cazul fiecărui proces de fabricare, operaţiile 1 (citirea parametrilor de intrare) şi 3 (scrierea soluţiei MIP în spaţiul de adrese) sunt bazate în principal pe nivelul serviciilor, operaţia 2 (rezolvarea

modelelor MIP) are loc la nivelul superior al arhitecturii iar operația 4 (fabricare piese) are loc la nivelul server-ului OPC UA şi al dispozitivelor de control ale stațiilor şi celulelor. Operația 1 are aceeaşi durată indiferent de numărul de piese comandate deoarece parametrii de intrare citiți din spațiul de adrese sunt independenți de comandă. Operația de scriere a soluției MIP în spațiul de adrese necesită timpi mai mari odată cu creşterea numărului de piese deoarece matricele înscrise în nodurile variabilă *Rezultate MIP* devin mai mari.

În figura 12 este prezentată soluția completă obținută prin aplicarea modelelor MIP în cazul în care se primeşte o comandă de nouă piese (trei din fiecare tip). Se poate observa faptul că atât duratele de procesare (tabel 1) cât şi timpii de transport (tabel 2) sunt respectați cu strictețe şi volumul de lucru este distribuit în mod echilibrat între cele patru stații şi zece celule.

REVENDICĂRI

1. Arhitectura propusă pentru optimizarea aplicațiilor industriale, **caracterizată prin aceea că** este formată din trei nivele: (1) server-ele OPC UA care modelează datele de la nivelul dispozitivelor şi astfel fiecare informație devine uşor accesibilă într-un mod unificat, (2) servicii software, care sunt organizate pe două nivele (servicii de bază şi servicii complexe) care au rolul de a asigura o comunicație mai uşoară între nivelul superior (CSP) şi nivelul inferior (OPC UA), în sensul că serviciile sunt folosite de modelele CSP atât pentru a citi datele de configurare stocate în cadrul variabilelor din spațiul de adrese al serverului OPC UA cât şi pentru a transporta soluția modelului CSP la serverul OPC UA astfel încât piesele să poate fi prelucrate automat, şi (3) un set de modele CSP precum şi un set de solvere care sunt folosite pentru rezolvarea acestor modele, determinându-se astfel soluții optime de prelucrare a comenzilor preluate.

2. Setul de algoritmi utilizați pentru generarea automată şi eficientă a spațiului de adrese al unui server OPC UA, **caracterizați prin aceea că**: (1) permit generarea rapidă a unor spații de adrese complexe pe baza datelor extrase dintr-o sursă de date (bază de date relațională, fişiere text, fişiere Excel), (2) pot genera noduri UA aparținând oricărei clase de noduri definite în specificația UA, şi (3) permit modificarea online sau offline a structurii spațiului de adrese, simplificând astfel activitățile de mentenanță la nivelul serverului OPC UA.

3. Nivelul de determinare a planurilor optime de fabricație, **caracterizat prin aceea că** poate utiliza orice tehnologie/tehnică de căutare a soluțiilor optime, nu neapărat metodologia bazată pe probleme de satisfacere a constrângerilor.

4. Servicii de bază utilizate la nivelul intermediar al arhitecturii, **caracterizate prin aceea că**: (1) permite realizarea operațiilor de bază cu serverul OPC UA (citire/modificare date, monitorizarea evenimentelor şi alarmelor, citirea datelor istorice procesate sau neprocesate), şi (2) pot fi reutilizate în orice aplicație a arhitecturii, inclusiv pentru generarea unor rapoarte pentru nivelul de management al întreprinderii.

5. Serviciile complexe utilizate pentru monitorizarea şi controlul proceselor de fabricație, **caracterizate prin aceea că**: (1) preiau soluția optimă de la un nivel superior al arhitecturii, (2) înscriu soluția optimă în serverul OPC UA, care comunică direct cu dispozitivele de control, (3) stabilesc o conexiune UA cu serverul UA pentru a monitoriza procesul de fabricație, şi (4) intervin în cazul în care apare o eroare şi fie opresc execuția procesului şi informează inginerul de apariția erorii, fie apelează un serviciu special utilizat pentru tratarea erorii.

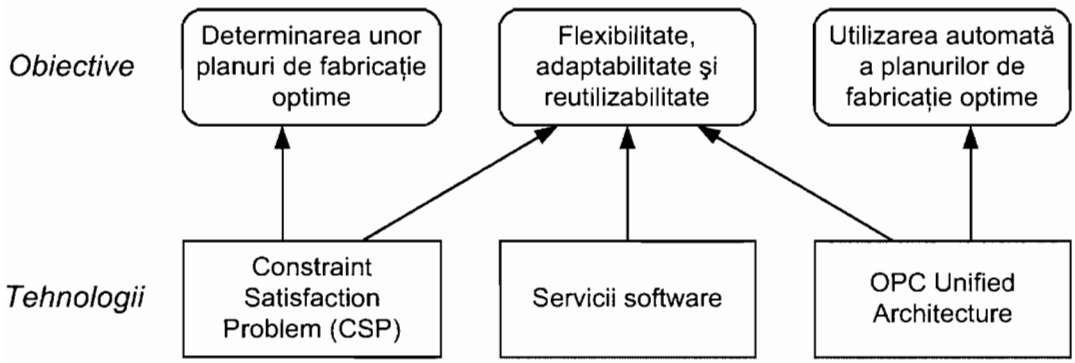

Fig. 1. Contribuția tehnologiilor la îndeplinirea obiectivelor arhitecturii/invenției

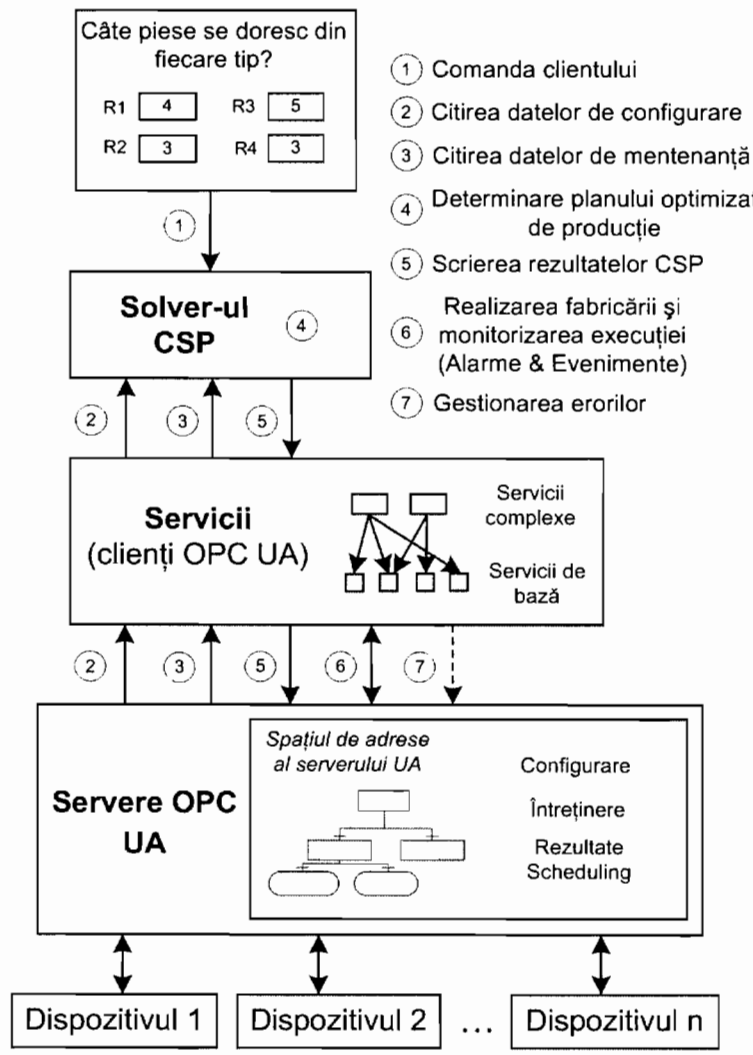

Fig. 2. Arhitectura propusă pentru optimizarea proceselor de fabricație

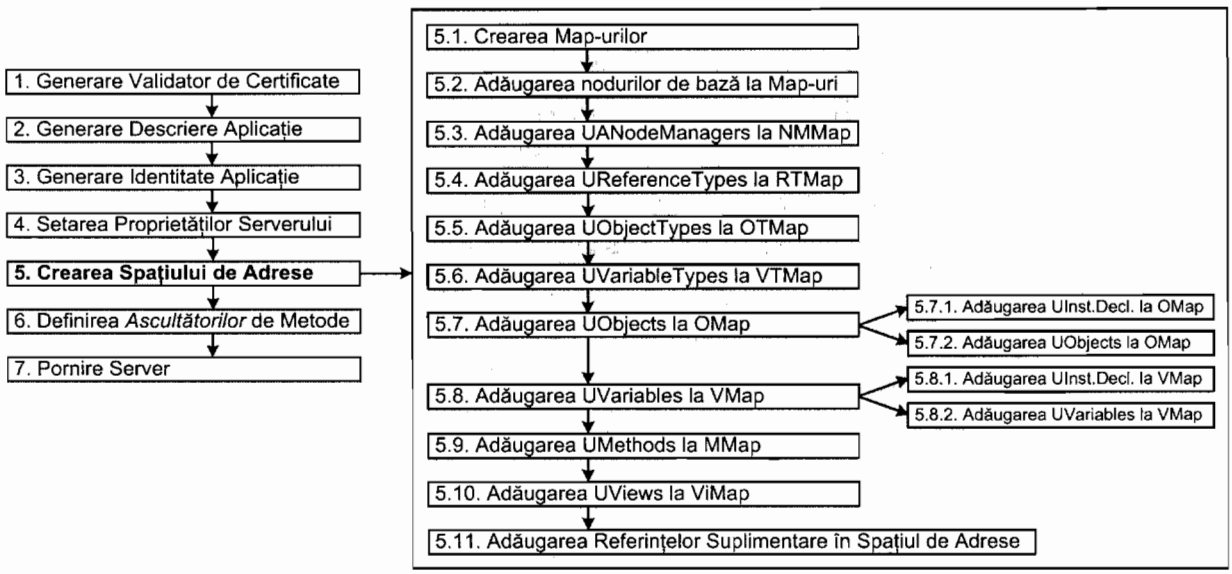

Fig. 3. Etapele de creare a unui server UA

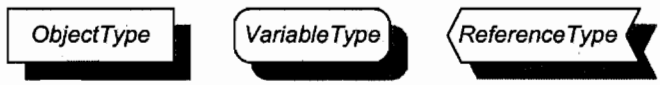

Fig. 4. Tipuri de noduri

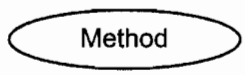

Fig. 5. Nod metodă

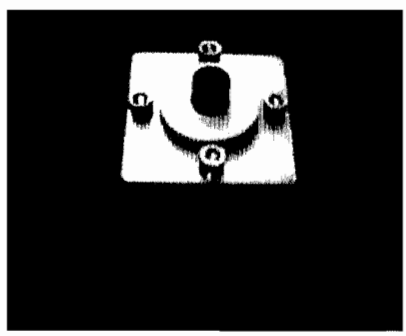

Fig. 6. Mecanism de rotație

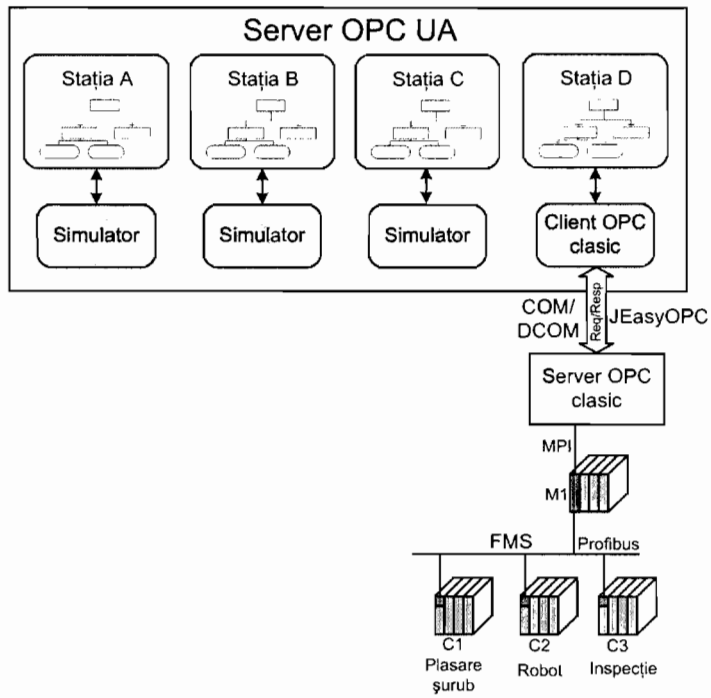

Fig. 7. Structura aplicației bazate pe linia flexibilă FMS 200

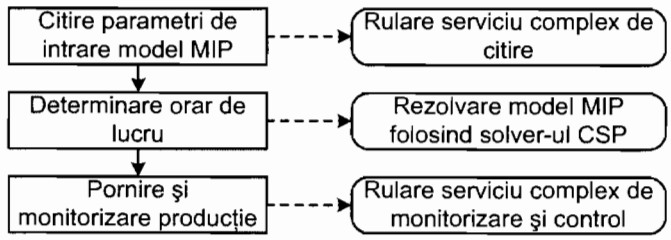

Fig. 8. Operații executate pentru fabricarea pieselor

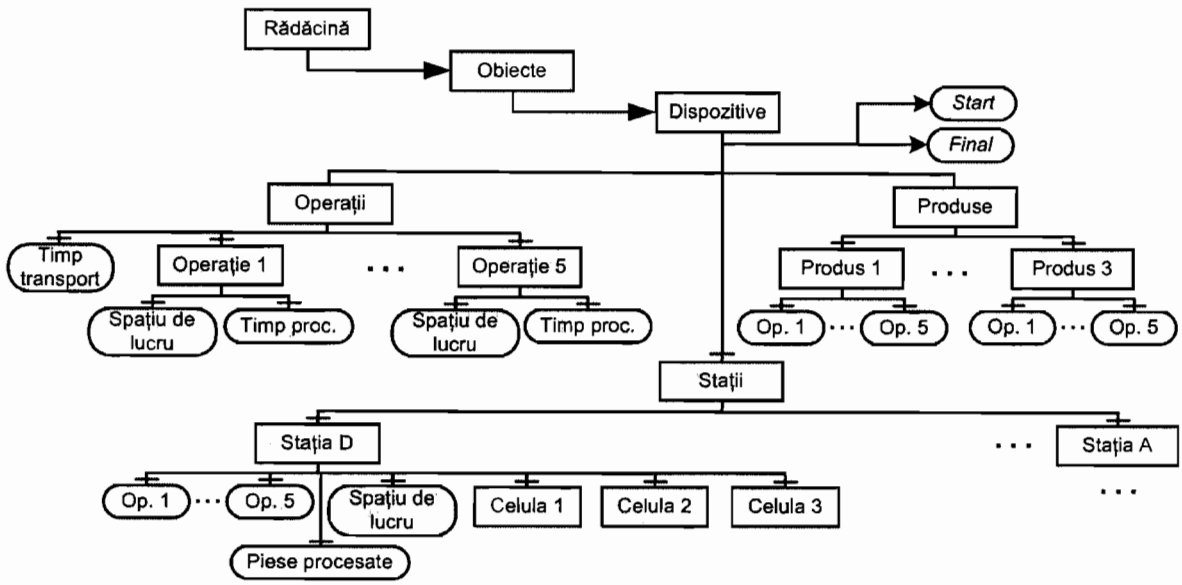

Fig. 9. Spațiul de adrese corespunzător aplicației

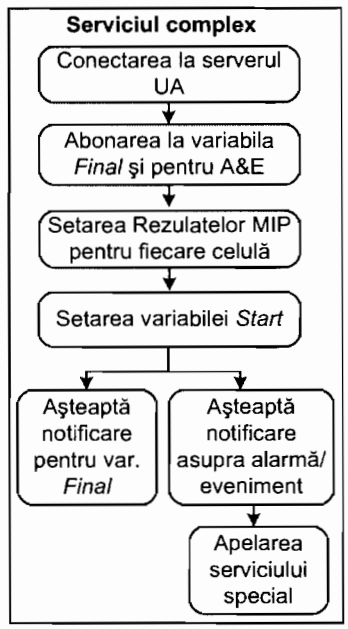

Fig. 10. Serviciul complex utilizat pentru monitorizarea şi controlul procesului de fabricare a mecanismelor de rotaţie

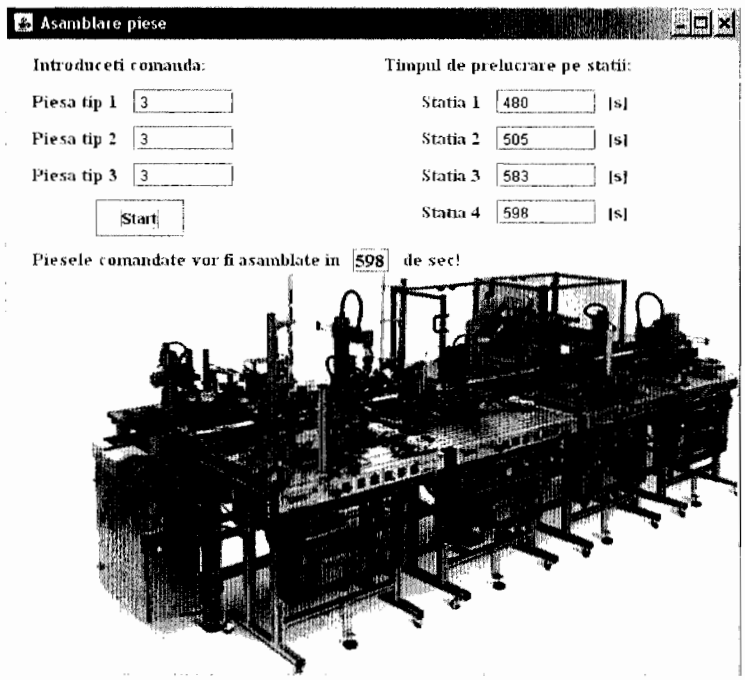

Fig. 11. Interfaţa grafică utilizată pentru plasarea unei comenzi

	Op. 1 → Stația 1: 375 - 435		Op. 1 → Stația 1: 281 - 341		Op. 1 → Stația 1: 196 - 256
	Op. 2 → Stația 1: 440 - 460		Op. 2 → Stația 1: 346 - 366		Op. 2 → Stația 1: 261 - 281
P1 Tip 1	Op. 3 → Stația 3: 480 - 495	**P2 Tip 1**	Op. 3 → Stația 3: 386 - 396	**P3 Tip 1**	Op. 3 → Stația 3: 317 - 327
	Op. 4 → Stația 3: 499 - 529		Op. 4 → Stația 3: 400 - 440		Op. 4 → Stația 3: 331 - 371
	Op. 5 → Stația 3: 534 - 544		Op. 5 → Stația 3: 445 - 455		Op. 5 → Stația 3: 376 - 386
	Op. 1 → Stația 1: 101 - 161		Op. 1 → Stația 1: 330 - 410		Op. 1 → Stația 1: 6 - 66
	Op. 2 → Stația 1: 166 - 196		Op. 2 → Stația 1: 415 - 435		Op. 2 → Stația 1: 71 - 101
P1 Tip 2	Op. 3 → Stația 3: 228 - 243	**P2 Tip 2**	Op. 3 → Stația 3: 455 - 470	**P3 Tip 2**	Op. 3 → Stația 3: 121 - 136
	Op. 4 → Stația 3: 247 - 297		Op. 4 → Stația 3: 474 - 524		Op. 4 → Stația 3: 140 - 190
	Op. 5 → Stația 3: 302 - 317		Op. 5 → Stația 3: 529 - 544		Op. 5 → Stația 3: 195 - 210
	Op. 1 → Stația 1: 110 - 190		Op. 1 → Stația 1: 220 - 300		Op. 1 → Stația 1: 0 - 80
	Op. 2 → Stația 1: 195 - 220		Op. 2 → Stația 1: 305 - 330		Op. 2 → Stația 1: 85 - 110
P1 Tip 3	Op. 3 → Stația 3: 262 - 287	**P2 Tip 3**	Op. 3 → Stația 3: 371 - 396	**P3 Tip 3**	Op. 3 → Stația 3: 153 - 178
	Op. 4 → Stația 3: 291 - 341		Op. 4 → Stația 3: 400 - 450		Op. 4 → Stația 3: 182 - 232
	Op. 5 → Stația 3: 346 - 371		Op. 5 → Stația 3: 455 - 480		Op. 5 → Stația 3: 237 - 262

Fig. 12. Orar de lucru pentru o comandă de 9 piese

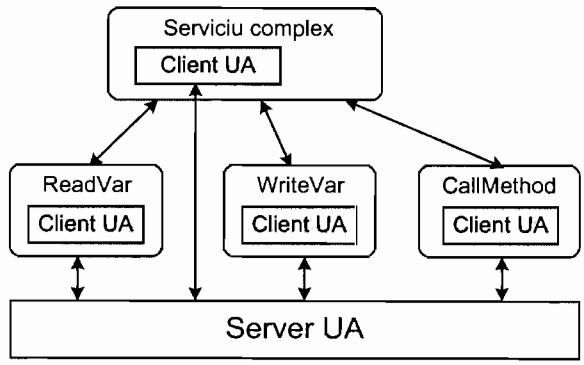

Fig. 13. Monitorizarea alarmelor şi a evenimentelor prin intermediul serviciilor