Mariusz PELC
POLITECHNIKA OPOLSKA, WEAiI,
ul. Sosnkowskiego 31, 45-272 OPOLE

Rekonfigurowalne autonomiczne systemy sterowania

Dr inż. Mariusz PELC

Jest obecnie pracownikiem Wydziału Elektrotechniki, Automatyki i Informatyki Politechniki Opolskiej. Jego działalność naukowa obejmuje nowoczesne komputerowe systemy sterowania, jak również klastry komputerowe i obliczenia równoległe. Najnowsza działalność badawcza dra inż. Mariusza Pelca prowadzona we współpracy z Uniwersytetem Greenwich w Londynie dotyczy systemów autonomicznych typu self-*, a w szczególności metody ich formalnej weryfikacji i walidacji.

e-mail: m.pelc@po.opole.pl

Streszczenie

W niniejszym artykule poruszona zostaje problematyka rekonfigurowalnych autonomicznych systemów sterowania działających bez (lub z możliwie niewielką) ingerencji człowieka, z drugiej jednak strony wyposażonych w mechanizm, który daje możliwość ich kontroli, a w sytuacji bardzo daleko posuniętych zmian środowiska (ang. *context awareness*), w którym te systemy operują, umożliwia ich szybką rekonfigurację. Jest to możliwe dzięki zaprezentowanej w tym artykule architekturze komponentu programowego, którego punkty decyzyjne (ang. *Decision Points*) stanowiące w istocie „logikę" tego komponentu, są otwarte (wymienne), to znaczy mogą być w zasadzie dowolnie kształtowane już po załadowaniu (uruchomieniu) tego komponentu w systemie docelowym. Architektura ta wspierana dedykowanym oprogramowaniem warstwy pośredniej (ang. *middleware*), umożliwia implementację efektywnego mechanizmu rekonfiguracji w oparciu o polityki.

Słowa kluczowe: komputerowe systemy sterowania, systemy autonomiczne, polityki.

Reconfigurable autonomic control systems

Abstract

In this paper a very relevant issue related to reconfigurable autonomic real-time control systems is undertaken. Designers of this kind of systems have to balance on the one side the ability of their operation with virtually no (or with minimum) human intervention, but on the other side, they have to implement a mechanism for easy and efficient reconfiguration in response to changing environmental conditions (context awareness). Although there are many techniques that may potentially be used to implement autonomic behaviour (such as Artificial Neural Networks, Fuzzy Logic, etc.) actually only policy-based computing seems to give the system designers enough freedom when it comes to specification of how the system should behave in response to the environmental changes. This ease results from the fact, that each policy is written using a Policy Description Language (PDL) which can be optimized for the given problem area (various control systems applications, business applications, etc.). As each PDL (for example AGILE PDL) offers besides a specific structure also a set of keywords strictly related to the problem domain, the system designers can easily express relations between context variables changes (these reflect the environment / context changes) and the requested changes to the system behaviour in a descriptive way. This is a very big advantage of policy-based computing over the alternative technologies. Policies themselves constitute the software component "logic" and because they (and thus the logic itself) can easily be replaced with newer (more optimized) versions, then in the result the same software component may behave completely different under the same environmental conditions. Policy-based computing is the ideal candidate technology to support reconfiguration of autonomic systems as this technology is not resource hungry, so it may be successfully applied to the embedded systems domain.

Keywords: computer control systems, policy-based computing.

1. Wprowadzenie

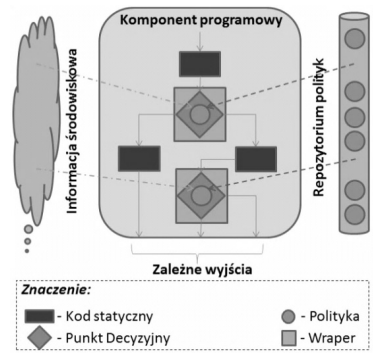
W dobie dość dynamicznego rozwoju komputerowych systemów sterowania coraz powszechniejsze staje się łączenie klasycznych algorytmów sterowania z technologiami, które z jednej strony pozwalają na zwiększenie autonomii tych systemów sterowania, a z drugiej strony pozwalają na ich szybką i efektywną rekonfigurację w czasie rzeczywistym. Dzięki temu klasyczne algorytmy sterowania zyskują zdolność strukturalnej adaptacji do zmian środowiska, a w efekcie pozwalają na zwiększenie elastyczności stosowanych dotychczas rozwiązań [1].

Pomimo istnienia wielu technologii, które wspierają projektowanie autonomicznych systemów sterowania (np. sztuczne sieci neuronowe, logika rozmyta, systemy neuronowo-rozmyte), jedynie niektóre z nich mogą być wykorzystane w systemach czasu rzeczywistego o ograniczonych zasobach (systemach wbudowanych). Ponadto, jeśli wziąć pod uwagę fakt, iż przy algorytmach ze sztucznymi sieciami neuronowymi, bądź też algorytmach rozmytych czy neuronowo-rozmytych, proces uzyskania nowej wersji „logiki" wymaga w praktyce ponownego, czasochłonnego uczenia sieci lub doboru / optymalizacji zmiennych lingwistycznych, wówczas okaże się, że wybór algorytmów komputerowych wykorzystujących polityki (ang. *policy-based computing*), gdzie „logika" formułowana jest w sposób opisowy (jak w przypadku systemów ekspertowych), może się okazać uzasadniony[2, 3].

W dalszej części tego artykułu zostanie zaprezentowana architektura rekonfigurowalnego komponentu programowego wraz z koherentnym oprogramowaniem warstwy pośredniej (rozdział 2). Następnie zostanie przedstawiona architektura klasycznego systemu sterowania rozbudowanego o elementy zapewniające autonomiczność / rekonfigurowalność (rozdział 3). W rozdziale przedstawiona zostanie struktura autonomicznego rekonfigurowalnego systemu sterowania. W rozdziale 4 pokrótce przedstawiony zostanie język opisu polityk AGILE, natomiast podsumowanie treści niniejszego artykułu wraz z opisem problemów otwartych znajdą się w rozdziałach 6 i 7.

2. Architektura komponentu z otwartymi punktami decyzyjnymi

Architektura komponentu z otwartymi punktami decyzyjnymi była wcześniej opisywana w [4]. Jej uproszczona wersja jest przedstawiona na rys. 1.



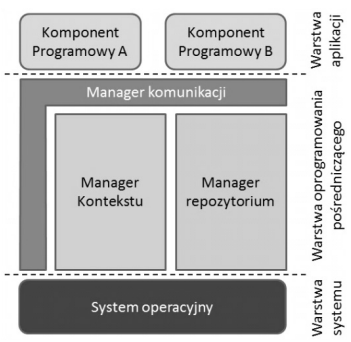
Rys. 1. Architektura komponentu z otwartymi punktami decyzyjnymi
Fig. 1. Open Decision Point Component Architecture

Architektura komponentu przedstawiona na rys. 1 zakłada, że każdy komponent programowy składa się z fragmentów kodu, które nie zawierają logiki (kod statyczny) oraz fragmentów kodu (Punktów Decyzyjnych – PD) decydujących o dalszym przebieg programu i ostatecznym wyniku przetwarzania. W każdym PD dochodzi do sprawdzenia aktualnego stanu środowiska (poprzez zmienne środowiskowe) i podjęcia decyzji w oparciu a aktualną politykę, jak komponent powinien się w danej sytuacji zachować, przy czym jako środowisko rozumiane jest wszystko to, co nie jest

zawarte lub wytworzone w obrębie samego komponentu; oznacza to, że źródłem informacji kontekstowej może być tak środowisko zewnętrzne dla systemu, w obrębie którego komponent operuje, jak i inne komponenty tego samego systemu. Ponieważ polityka jest elementem, który może być zmieniony w dowolnym momencie (także po uruchomieniu komponentu w systemie docelowym), zatem ten sam komponent (a w szerszym kontekście – cały system) może się zachowywać różnie dla takich samych warunkach środowiskowych. Polityki mogą być składowane w repozytorium (ang. *policy repository*), którym może być np. dysk twardy, karta SD czy karta dołączona poprzez port USB (typowe dla systemów wbudowanych).

Niezwykle istotnym elementem architektury komponentu z otwartymi PD jest *Wraper*. Element ten zapewnia stabilne zachowanie komponentu w sytuacji, gdy z jakiegoś powodu przetwarzana polityka zawiedzie i podjęta decyzja będzie błędna, na przykład w sytuacji, gdy nie są dostępne wszystkie zmienne środowiskowe potrzebne do podjęcia określonej decyzji. W takiej sytuacji *Wraper* na podstawie analizy wyniku zwracanego po przetworzeniu polityki może go odrzucić i zwrócić wcześniej ustalony, „statyczny" i bezpieczny wynik.

Oczywiście, aby opisany powyżej komponent mógł poprawnie działać w obrębie docelowego systemu uruchomieniowego, system ten musi wspierać zarówno zarządzanie informacją o środowisku (informacją kontekstową), jak i zarządzanie repozytorium polityk. Źródła informacji kontekstowej powinny być rejestrowane, rejestrowani powinni być również odbiorcy informacji kontekstowej. Podobnie, jako że różne punkty decyzyjne będą potrzebowały różnych polityk, musi istnieć serwis systemowy, który będzie dostarczał właściwe polityki do właściwych komponentów, z uwzględnieniem ich wersji (celem np. aktualizacji, gdy pojawi się nowsza wersja polityki załadowana do któregoś punktu decyzyjnego), pochodzenia (można wykluczyć w ten sposób polityki pochodzące z niezaufanych źródeł), itp. Dodatkowo, aby cały ten system komunikacyjny mógł działać, konieczna jest implementacja dedykowanego serwisu, który będzie odpowiadał za przesyłanie komunikatów pomiędzy elementami systemu związanymi z obsługą informacji kontekstowej oraz polityk. Serwis ten będzie rejestrował źródła oraz odbiorców informacji kontekstowej, będzie zapewniał jej dostarczanie oraz aktualizację do właściwych komponentów, jak również będzie przekazywał właściwe polityki do właściwych punktów decyzyjnych.

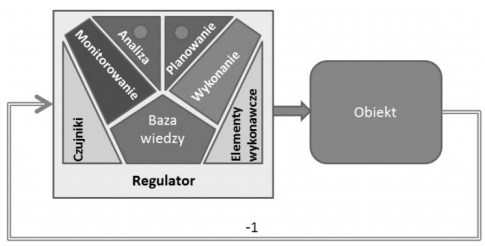


Rys. 2. Uproszczona architektura systemu z oprogramowaniem pośredniczącym
Fig. 2. Simplified architecture of a system with the middleware

Ponieważ trudno oczekiwać, że każdy potencjalny system docelowy będzie wyposażony we wszelkie potrzebne serwisy wspierające opisywaną architekturę komponentu programowego, najsensowniejsze jest dostarczanie gotowego, kompleksowego rozwiązania, które obejmie również oprogramowanie warstwy pośredniczącej (ang. *middleware*) [5]. Jeśli to właśnie oprogramowanie warstwy pośredniczącej będzie dostarczało wszystkich niezbędnych serwisów, sam system uruchomieniowy (operacyjny) powinien dostarczać jedynie API (ang. *Application Programming Interface*) do komunikacji z oprogramowaniem warstwy pośredniczącej. Ogólną hierarchiczna architektura systemu z oprogramowaniem warstwy pośredniczącej przedstawiono na rys. 2.

3. Rekonfigurowalny autonomiczny system sterowania

Realizacja autonomicznego rekonfigurowalnego systemu sterowania może wykorzystywać zmodyfikowaną architekturę *MAPE* (ang. *Monitor->Analyze->Plan->Execute*) [6]. Architektura ta jest dość powszechnie wykorzystywana w systemach autonomicznych. Tak więc naturalnym rozwiązaniem staje się wykorzystanie tej architektury także w obrębie struktury rekonfigurowalnego autonomicznego systemu sterowania opisywanego w tym rozdziale, co jest widoczne na rys. 3.



Rys. 3. Struktura rekonfigurowalnego autonomicznego systemu sterowania
Fig. 3. Structure of the reconfigurable autonomic control system

W przedstawionej na rys. 3 strukturze autonomicznego rekonfigurowalnego układu sterowania można zauważyć, że standardowy regulator został zastąpiony bardziej rozbudowaną wersją, która implementuje architekturę *MAPE*, odpowiedzialną za autonomiczne zachowanie. Z drugiej strony można zauważyć, że moduły odpowiedzialne za fazę analizy i planowania, mogą implementować wewnętrznie architekturę z otwartymi punktami decyzyjnymi. Oznacza to, że cała architektura *MAPE* uzyskuje w ten sposób zdolność do rekonfiguracji, bowiem strategiczne decyzje dotyczące analizy i planowania mogą być konfigurowane za pomocą polityk. W zależności od wersji tych polityk ten sam regulator może generować zupełnie inne sterowania w tych samych warunków środowiskowych.

Warto dodać, iż prezentowana struktura systemu sterowania jest całkowicie uniwersalna i pozwala na realizację bardzo szerokiej gamy algorytmów sterowania. Można ją również stosować w systemach czasu rzeczywistego, jednak w tym przypadku konieczne jest wykonanie badań celem ustalenia nakładów czasowych potrzebnych do wykonania pełnego cyklu decyzyjnego [9].

4. Język opisu polityk AGILE

To, co stanowi o bardzo dużej elastyczności systemów wykorzystujących polityki to sposób opisu / zapisu algorytmu decyzyjnego. Polityki są zwykle opisywane za pomocą tzw. języków opisu polityk (ang. *Policy Description Languages*). Jednym z takich języków, choć nie jedynym (przykład innego formalnego języka opisu politk można znaleźć np. w [10]), jest język AGILE, który jest oparty na języku XML (ma to istotne znaczenie ze względu na popularność języka XML) i który został opisany po raz pierwszy w [7]. Język AGILE umożliwia wydzielenie w tekście / treści polityki pewne wyspecjalizowne bloki funkcjonalne, m.in.:

- *Action* - blok zawierający zestaw instrukcji (akcji) do wykonania,
- *Rule* - blok zawierający proste lub złożone reguły decyzyjne (w zależności od tego, która reguła jest spełniona, wywoływana jest określona akcja),
- *ToleranceRangeCheck* - blok uruchamiający określone akcje w zależności od tego, jaka jest relacja pomiędzy wielkością kontrolowaną a wielkością odniesienia,
- *UtilityFunction* - blok pozwalający na uruchamianie określonych akcji w zależności od wartości funkcji użyteczności,
- *Template* - blok konfigurujący politykę.

W języku AGILE informacja o środowisku jest przekazywana za pomocą zmiennych środowiskowych, natomiast wynik działania polityki może być zwrócony bezpośrednio (strukturalnie), lub pośrednio za pomocą specjalnych zmiennych wyjściowych.

5. Przykładowa polityka w języku AGILE

Najlepszą ilustracją elastyczności języka AGILE może być poniższy przykład polityki, która decyduje o parametrach pracy regulatora typu PI (patrz rys. 4).

```
<!-- Policy Definition XML file: Policy Language version 1.2 -->
<!-- Application: PI Control -->
<PolicySuite PolicyType="PI Controller Tuning">
  <EnvironmentVariables>
    <EVariable Name="Epsilon" Type="real"/>
  </EnvironmentVariables>
  <InternalVariables>
    <IVariable Name="EpsRef" Type="real"/>
  </InternalVariables>
  <OutputVariables>
    <OVariable Name="Kp" Type="real"/>
    <OVariable Name="Ti" Type="real"/>
  </OutputVariales>
  <Templates>
    <Template Name="T1">
      <Assign Variable="EpsRef" Value="50"/>
    </Template>
  </Templates>
  <ReturnValues>
    <ReturnValue Name="StatusOK" Value="0"/>
    <ReturnValue Name="StatusNOK" Value="1"/>
  </ReturnValues>
  <Actions>
    <Action Name="Start">
      <EvaluateRule Rule="EpsCheck"/>
      <Return ReturnValue="StatusNOK"/>
    </Acton>
    <Action Name="SetPar1">
      <Assign LHS="Kp" RHS="5"/>
      <Assign LHS="Ti" RHS="5"/>
      <Return ReturnValue="StatusOK"/>
    </Action>
    <Action Name="SetPar2">
      <Assign LHS="Kp" RHS="3"/>
      <Assign LHS="Ti" RHS="3"/>
      <Return ReturnValue="StatusOK"/>
    </Action>
  </Actions>
  <Rules>
    <Rule Name="EpsCheck" LHS="Epsilon" Op="LT"
      RHS="EpsRef" ActionIfTrue="SetPar1"
      ElseAction="SetPar2"/>
  <Rules/>
  <Policies>
    <Policy Name="Policy1" PolicyType="NormalPolicy">
      <Load Template="T1"/>
      <Execute Action="Start"/>
    </Policy>
  </Policies>
</PolicySuite>
```

Rys. 4. Przykładowa polityka określająca parametry regulatora PI
Fig. 4. Exemplar policy determining PI controller parameters

W powyższym przykładzie polityki w zależności od tego, jaka jest relacja pomiędzy aktualną wartością uchybu regulacji i wartością odniesienia dla uchybu regulacji (reguła *EpsCheck*), parametry regulatora (wzmocnienie i stała całkowania) zostają albo zmniejszone, albo zwiększone, dzięki czemu zmianie ulega transmitancja regulatora PI określona zależnością (1) może być dostosowana do aktualnych uwarunkowań środowiskowych.

$$G(s) = K_p\left(1 + \frac{1}{T_i s}\right), \tag{1}$$

6. Wnioski

W niniejszym artykule zaprezentowana została architektura komponentu programowego oraz oprogramowania warstwy pośredniczącej wspierająca rekonfigurację oraz autonomiczność komputerowych systemów sterowania. Architektura takiego komponentu może przynieść szczególnie wiele korzyści w połączeniu z architekturą systemu *MAPE*, dzięki czemu system sterowania zyskuje również cechy autonomiczności.

Rekonfiguracja wspomnianego systemu sterowania może być dokonana poprzez zmianę polityk w czasie rzeczywistym. Logika zawarta w polityce pozwala podejmować strategiczne decyzje, w tym np. dotyczące istotnych parametrów systemu sterowania. Jako przykład zaprezentowana została polityka doboru parametrów regulatora PI.

Część rozwiązań opisanych w tym artykule znalazło zastosowanie jako element projektu *DySCAS* [8, 11] finansowanego ze środków UE.

7. Problemy otwarte

Choć przedstawione rozwiązania wydają się być ciekawą odpowiedzią na wiele istotnych niedogodności wynikających ze stosowania klasycznych algorytmów sterowania (chodzi głównie o ich zamkniętą, mało elastyczną strukturę) w nowoczesnych komputerowych systemach sterowania, są one również źródłem innych problemów. Do najistotniejszych należy oczywiście walidacja takiego hybrydowego / hierarchicznego systemu sterowania. W przypadku klasycznych algorytmów można w sposób analityczny określić np. stabilność, czy odporność systemu, jak dotąd jednak nie opracowano żadnej metody formalnej pozwalające na tego typu analizę w odniesieniu do logiki zawartej w politykach AGILE. Prace nad opracowaniem tego typu metodologii trwają, a wyniki tych prac zostaną opublikowane w nieodległej przyszłości.

8. Literatura

[1] Pelc M., Anthony R., Byrski W.: Policy Supervised Exact State Reconstruction in Real-Time Embedded Control Systems. Proceedings of ACD2009, Zielona Góra, 2009.
[2] Anthony R., Pelc M., Ward P., Hawthorne J., Pelc M.: A Flexible and Robust Run-Time Configuration for Self-Managing Systemss. Proceedings of SASO, Isola di San Servolo (Venice), Italy 2008.
[3] Anthony R., Pelc M., Ward P., Hawthorne J.: A Run-Time Configurable Software Architecture for Self-Managing Systems. Proceedings of ICAC 2008, Chicago, Illinois, USA, 2008.
[4] Anthony R., Pelc M., Ward P., Hawthorne J.: A Software Architecture supporting Run-Time Configuration and Self-Management. Communications of SIWN Journal, Vol. 7, May 2009, pp. 103-112.
[5] Pelc M., Anthony R., Ward P., Hawthorne J.: Practical Implementation of a Middleware and Software Component Architecture supporting Reconfigurability of Real-Time Embedded Systems. Proceedings of 2009 International Conference on Computational Science and Engineering, Vancouver, Canada, pp. 394-401.
[6] Anthony R., Pelc M., Byrski W.: Context-Aware Reconfiguration of Autonomic Managers in Real-Time Control Applicaitons. Proceedings of ICAC2010, Washington DC, USA, 2010.
[7] Anthony R.: A policy-definition language and prototype implementation library for policy-based autonomic systems. Proceedings of 3rd International Conference on Autonomic Computing (ICAC2006), pages 265–276, 2006.
[8] Anthony R., DeJiu Chen, Pelc M., Persson M., T¨orngren M., Ward P., Hawthorne J.: Context-Aware Adaptation in DySCAS. Electronic Communications of the EASST Journal, 2009, Volume 19, s. 1-15.
[9] Pelc M., Anthony R., Ward P., Hawthorne J.: Towards Context-Awareness and Self-Configuration of Embedded Systems: Resource Usage and Real-Time Characteristics. Proceedings of ACD2009, Zielona Góra, 2009.
[10] Koch, T., Krell C., Kramer B.: Policy definition language for automated management of distributed systems, Proceedings of IEEE Second International Workshop on, Toronto, Ont. , Canada, s. 55-64.
[11] http://www.dyscas.org

otrzymano / received: 21.07.2010
przyjęto do druku / accepted: 04.04.2011 *artykuł recenzowany*