Realizacja mechanizmu lokalnej wymiany informacji w ramach platformy komunikacyjnej w rozproszonych systemach sterowania

Przemysław Strzelczyk, Krzysztof Tomczewski
Politechnika Opolska, Wydział Elektrotechniki, Automatyki i Informatyki, ul. Prószkowska 76 (bud. 1), 45-758 Opole

Streszczenie: W artykule przedstawiono sposób realizacji lokalnego mechanizmu komunikacji w generycznej platformie programowej realizującej wymianę danych, przystosowanej do pracy w rozproszonych systemach sterowania. W pracy omówiono strukturę warstw oprogramowania odpowiedzialną za transport pakietów wewnątrz pojedynczej instancji platformy. Zaprezentowano przykładowe scenariusze użycia mechanizmów komunikacji lokalnej.

Słowa kluczowe: rozproszone systemy sterowania, platforma komunikacyjna, wymiany danych

1. Wprowadzenie

Jednym z podstawowych zagadnień koniecznych do realizacji w rozproszonych systemach sterowania jest zapewnienie szybkiej i poprawnej komunikacji między jej węzłami. W pracy zaprezentowana została platforma informatyczna przystosowana do realizacji tego celu. Umożliwia ona uzyskanie wysokiej wydajności przez udostępnienie mechanizmów pozwalających na przesyłanie informacji zarówno lokalnie – w obrębie pojedynczego węzła, jak i w całej sieci między jej węzłami. W artykule przedstawiono mechanizm lokalnej wymiany informacji. Wydzielenie komunikacji lokalnej w obrębie pojedynczej struktury sprzętowej z ogólnego systemu wymiany informacji powoduje odciążenie łącz komunikacyjnych i umożliwia budowę układów o większej liczbie węzłów. Oprogramowanie platformy zostało przystosowane do pracy i uruchomione z systemem operacyjnym Linux Debian.

Głównym celem projektu jest opracowanie oprogramowania pełniącego rolę pośrednika między systemem operacyjnym a aplikacjami sterującymi. Każda z aplikacji uruchomionych na platformie może wykorzystywać jej funkcjonalności przez odwoływanie się do funkcji API. Program opracowany do współpracy z platformą komunikacyjną nie wymaga większych zmian w celu przystosowania do współpracy z innym systemem operacyjnym oraz z inną platformą sprzętową. Każda aplikacja może realizować dedykowane dla niej zadania, jednocześnie umożliwiając wymianę informacji z innymi programami uruchomionymi na platformie w obrębie danego węzła, jak i z aplikacjami uruchomionymi w innych węzłach systemu. Zadaniem platformy jest pełnienie roli pośrednika między aplikacjami a systemem operacyjnym, przy zapewnieniu generyczności kodu programów bazujących na niej. Zastosowanie platformy pozwala na opracowanie oprogramowania dedykowanego dla rozproszonych systemów sterowania. Najważniejszym atutem tego rozwiązania jest możliwość wykreowania środowiska, które pozwala na przyłączenie dużej liczby modułów i udostępnienie mechanizmów komunikacyjnych bez podziału na wymianę globalną i lokalną. Oznacza to, że projektując aplikację, stosuje się te same funkcje API zarówno w przypadku przesłania informacji między aplikacjami uruchomionymi w tym samym węźle, jak i w węzłach znajdujących się w innych punktach układu sterowania. Program wykorzystujący udostępnioną przez platformę bibliotekę traktuje aplikacje uruchomione w różnych węzłach tak, jakby były one uruchomione lokalnie w tym samym węźle układu sterowania.

Jednym z najbardziej popularnych już istniejących rozwiązań, umożliwiających lokalną, jak i globalną komunikację w systemach sterowania, jest biblioteka OpenRTM, bazująca na wyspecyfikowanych standardach RT-middleware (ang. *Robotics Technology Middleware*). [7] Główną różnicą między opisywanym rozwiązaniem a istniejącym oprogramowaniem OpenRTM jest to, że prezentowana platforma umożliwia łatwiejszą integrację już istniejących systemów – brak tu ustalonych reguł odnoszących się do struktury programu. Proponowana platforma komunikacyjna została pozbawiona wszelkich zbędnych elementów umożliwiających blokowe tworzenie oprogramowania ze względu na czas wykonywania oraz kompilacji projektu. Nie zawiera ona żadnych reguł dotyczących procesu sterowania. Reguły te definiuje dopiero programista tworzący końcową aplikację.

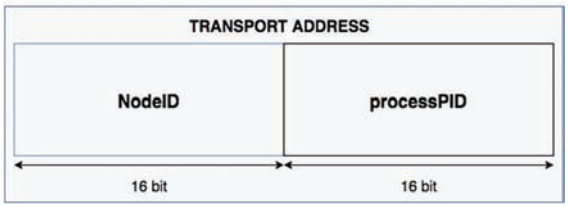
2. Mechanizmy wymiany danych

Zadaniem platformy programowej jest udostępnienie mechanizmów umożliwiających twórcom aplikacji sterujących realizację mechanizmów wymiany danych w systemach sterowania o strukturze rozproszonej.

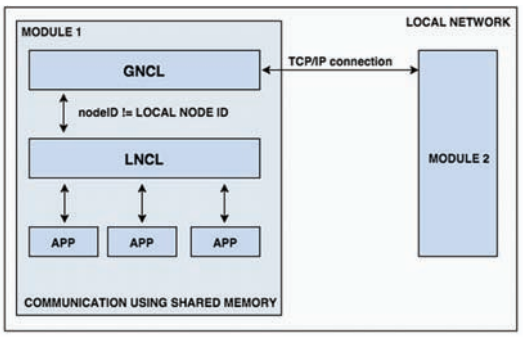
Funkcje udostępniane przez główną bibliotekę prezentowanej platformy NodeAPI oparte są ściśle na wykorzystaniu systemowych mechanizmów wymiany danych. Środowisko to może

Autor korespondujący:
Przemysław Strzelczyk, przemyslawstrzelczyk@gmail.com

Artykuł recenzowany
nadesłany 9.10.2015, przyjęty do druku 3.02.2016



Rys. 1. Budowa adresu transportowego (ang. Transport ID)
Fig. 1. Transport ID structure



Rys. 2. Routing danych między węzłami platformy
Fig. 2. Data flow between platform nodes

zostać zainstalowane na dowolnej platformie sprzętowej, po dołączeniu odpowiednich bibliotek. Do przeprowadzenia testów wybrano system Linux Debian wraz z niezbędnymi narzędziami służącymi do kompilacji kodu platformy, jednak po dostosowaniu może ona współpracować z dowolnym systemem operacyjnym. Kod platformy zawiera elementy języka C++ wykorzystujące funkcjonalności wersji C++11. Z tego powodu wymagana jest kompilacja z udziałem kompilatora gcc, który wspiera tę wersję. Preferowana wersja to 5.11 ze względu na pełne wsparcie nowego standardu języka. Pliki konfigurujące poprawną kompilację (ang. *makefiles*) są generowane za pomocą CMake w wersji 2.8.10 [1]. System Linux Debian dostarcza szereg mechanizmów komunikacji wewnątrzsystemowej: łącza (potoki) nazwane i nienazwane, pamięć współdzieloną, kolejki itp. Do realizacji mechanizmów wymiany danych wewnątrz pojedynczej platformy sprzętowej wybrano gniazda dziedziny Linuxa (ang. *Linux domain sockets*). Gniazda te wymieniają informacje za pomocą pamięci współdzielonej. Zaletą wykorzystania takiego rozwiązania jest prostota implementacji. Interfejs funkcyjny odpowiada stosowanemu w przypadku normalnych gniazd, jakie są tworzone dla pakietów TCP/IP lub UDP. Umożliwia on tworzenie funkcjonalności lokalnej wymiany danych [2].

3. Warstwa Local Node Communication Layer

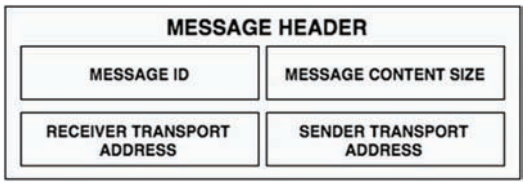
Najważniejszym elementem mechanizmu wymiany informacji jest warstwa LNCL (ang. *Local Node Communication Layer*). Warstwa ta pośredniczy podczas próby przesyłania danych w obrębie jednej platformy sprzętowej. Do głównych zadań tej warstwy należą:

- routing danych w zakresie lokalnym,
- pośrednictwo w globalnym routingu danych,
- rejestracja, przydzielanie i derejestracja adresów transportowych,
- gromadzenie informacji na temat aplikacji korzystających z usług przesyłu danych.

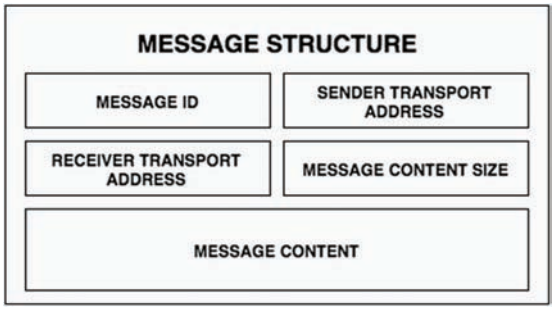
W takcie sekwencji uruchomieniowej warstwa LNCL ma najwyższy priorytet. Oznacza to, że jest ona pierwszą warstwą programową, która zostanie uruchomiona. Kolejność uruchamiania oprogramowania jest istotna, ponieważ działanie niektórych elementów jest niezbędne do załączenia kolejnych. Sekwencja uruchomieniowa realizowana jest za pomocą klasy z wbudowaną funkcjonalnością pozwalającą na zapis kolejności uruchomieniowej w postaci drzewa zależności. [3] Funkcjonalność ta umożliwia szybką modyfikację oraz odczyt utworzonej ścieżki. Głównym założeniem mechanizmu transportowego platformy jest udostępnianie uruchomionym aplikacjom sterującym i monitorującym rejestracji własnego adresu transportowego. Dzięki pozyskanemu adresowi aplikacja otrzymuje unikalny identyfikator, zarówno w obrębie lokalnym, jak i globalnym. Rejestracja wspomnianego adresu odbywa się przez użycie funkcji *RegisterTransportAddress*. Poprawne wykonanie funkcji powoduje, że system przydziela 32-bitowy adres transportowy o strukturze przedstawionej na rys. 1.

Pierwsze przesyłane 16 bitów TID (ang. *Transport ID*) określa adres węzła platformy (modułu sprzętowego), pozostałych 16 oznacza adres aplikacji. W ramach komunikacji lokalnej adres węzła platformy nie ma znaczenia. Może zostać uzupełniony zerami, jeśli aplikacje komunikują się w obrębie jednego węzła sprzętowego. LNCL zareaguje tylko wówczas, gdy ta część adresu będzie różna od zera. W takim przypadku warstwa ta sprawdzi, czy adresowany węzeł istnieje w systemie, wykorzystując w tym celu odpowiednie zapytanie do warstwy NDL (ang. *Nodes Detection Layer*). Warstwa NDL udostępnia funkcjonalność umożliwiającą pozyskanie listy węzłów dostępnych w ramach całego rozproszonego systemu sterowania. Jeśli dany węzeł istnieje, to warstwa LNCL przekaże dane do warstwy GNCL (ang. *Global Node Communication Layer*) w celu wysłania ich do innego modułu sprzętowego.

W związku z tym, że nie ma żadnych ograniczeń co do wielkości pakietów przesyłanych za pośrednictwem warstwy LNCL, przesyłanie danych zostało podzielone na dwie części. W chwili, gdy jedna aplikacja, po poprawnej rejestracji adresu transportowego, zamierza wysłać dane do drugiej już zarejestrowanej aplikacji, programista powinien wywołać funkcję *CreateMessage*. Funkcja ta umożliwia przygotowanie nagłówka wiadomości docelowej z danymi do wysyłki oraz wysłanie go do warstwy LNCL. Nagłówek informuje warstwę, że mechanizmy w niej zaimplementowane powinny oczekiwać nadejścia pakietu o podanych w nagłówku rozmiarach. Po wysłaniu nagłówek trafia do części LNCL odpowiedzialnej za obsługę ruchu sieciowego, która w trybie ciągłym odbiera struktury nagłówków przekazane do tej warstwy. Przesłana część zostaje sprawdzona pod kątem poprawności, a wydobyte dane zostają przekazane do elementu zarządzającego obsługą wiadomości (ang. *Message Dispatcher*). Taka organizacja komunikacji sprawia, że opracowana warstwa pośrednicząca umożliwia wymianę danych o dowolnej strukturze i wielkości, dzięki czemu w systemie funkcjonować mogą jednocześnie aplikacje sterujące i monitorujące, wykorzystujące dane zapisane w różnych formatach. Implementacja tego elementu warstwy została usytuowana w klasie *Dispatcher*. LNCL wykorzystuje tę klasę przez utworzenie jej obiektu, a następnie wywołanie metody *handleMessage*. Metoda ta przejmuje informacje wydobyte z nagłówka wiadomości, jako swoje parametry wejściowe, a następnie wyszukuje odpowiednią metodę obsługi danego zdarzenia (ang. *event handler*). Obiekty tej klasy, zawierające metody obsługi zdarzenia, mogą zostać dodane do elementu przez wywołanie funkcji *addHandler*. Obiekt taki powinien być utworzony za pośrednictwem klasy, która dziedziczy po klasie abstrakcyjnej *IHandler*. Dziedziczenie jest wymogiem



Rys. 3. Budowa nagłówka wiadomości LNCL
Fig. 3. LNCL message header structure



Rys. 4. Budowa ramki wiadomości
Fig. 4. Message frame structure

elementu zarządzającego obsługą wiadomości, ponieważ umożliwia to założenie, że klasa na podstawie której został utworzony obiekt, zawiera zaimplementowaną metodę obsługi zdarzenia. Zaimplementowanie metody jest wymuszane zastosowaniem czysto wirtualnych metod (ang. *pure virtual metods*) wewnątrz abstrakcyjnej klasy bazowej *IHandler*. [4] Na obecnym etapie realizacji środowiska, element *Message Dispatcher* jest odpowiedzialny za rozpoznawanie następujących typów wiadomości:
— typ: 0x1FF – rejestracja w systemie adresu transportowego,
— typ: 0x3FF – usunięcie z systemu adresu transportowego.

Przed rozpoczęciem procesu wyszukiwania metod obsługi następuje sprawdzenie, czy przesłaną strukturę oznaczono jako jeden z dwóch wymienionych typów. Jeśli struktura nie została oznaczona, to *Message Dispatcher* sprawdza, czy nadawca wiadomości istnieje, a następnie stosuje metodę obsługującą przekazywanie wiadomości do odbiorcy (ang. *forward message event handler*).

Po analizie i poprawnej weryfikacji nagłówka, wiadomość zostaje przekazana do odbiorcy, przy czym przyjęto założenie, że odbiorca wiadomości zna rozmiar przychodzących do niego wiadomości.

4. Platforma testowa

Po zrealizowaniu wszystkich opisanych założeń i opracowaniu oprogramowania, skonfigurowano platformę testową składającą się z dwóch modułów Raspberry PI w wersji 2, połączonych ze sobą przewodowo, z wykorzystaniem routera firmy Mikrotik 750G. W obu modułach zainstalowano system Linux Debian. Następnie skonfigurowano ustawienia elementu systemu operacyjnego *systemd* odpowiadającego za uruchamianie jego funkcjonalności, biorąc pod uwagę elementy niezbędne do uruchomienia platformy. Jednym z kluczowych elementów, który powinien zostać obsłużony i poprawnie uruchomiony przez system przed uruchomieniem oprogramowania platformy, jest element obsługi sieci przewodowej. Działanie środowiska bez funkcjonalności wymiany informacji w sieci ograniczałoby się jedynie do wymiany informacji lokalnej za pośrednictwem LNCL. Rozwiązanie przewodowe zapewnia krótszy czas wymiany danych. Docelowo oprogramowanie platformy będzie umożliwiało wymianę informacji, zarówno za pośrednictwem sieci przewodowej jak i bezprzewodowej. Poprawność komunikacji lokalnej została sprawdzona przez uruchomienie dwóch programów wykorzystujących bibliotekę platformy NodeAPI. Jeden z nich miał zaimplementowany mechanizm wysyłania danych a drugi ich odbierania. Test został przeprowadzony dla danych o różnych rozmiarach. Program wysyłający dane w każdym pakiecie przesyłał informację o rozmiarze kolejnego pakietu. Dzięki temu program odbierający pakiety mógł odbierać pakiety o zmiennej długości.

5. Podsumowanie

Przedstawiony sposób implementacji lokalnej wymiany danych w obrębie pojedynczego węzła sprzętowego platformy komunikacyjnej cechuje się prostotą użycia i krótkim czasem przesyłania danych. Umożliwiło to wykorzystanie pamięci współdzielonej. Konstrukcja warstwy LNCL pozwala na rozszerzenie oferowanych funkcjonalności o kolejne, dzięki zaimplementowanemu mechanizmowi generycznej obsługi zadań (*Message Dispatcher*), bez zmiany już zaimplementowanych algorytmów. Prezentowana platforma komunikacyjna w wersji docelowej powinna umożliwić m.in. realizację systemów typu SCADA (ang. *Supervisory Control And Data Acquisition*) oraz innych zajmujących się kolekcjonowaniem i przetwarzaniem zgromadzonych danych z odległych od siebie węzłów sieci. Opisane rozwiązanie w szczególności nadaje się do zastosowania w rozbudowanych systemach robotyki mobilnej, np. do sterowania rojem robotów, gdzie szybka i stabilna komunikacja w obrębie wszystkich jednostek i między nimi jest bardzo istotna.

Bibliografia

1. Wojtczyk M., Knoll A, *A Cross Platform Development Workflow for C/C++ Applications*, The Third International Conference on Software Engineering Advances (ICSEA-2008), 224–229, DOI: 10.1109/ICSEA.2008.41.
2. Shapley Gray J., *Interprocess Communications in Linux*, ISBN: 0-13-046042-7, Prentice Hall Professional 2003.
3. Cormen H.T., Leiserson E.C., Rivest L.R, Stein C., *Introduction to Algorithms*, ISBN: 0-262-03384-4, Prentice Hall Professional 2009.
4. Stroustrup B., *Język C++. Kompedium wiedzy*, ISBN: 978-83-246-8530-1, Helion 2014.
5. Itami Y., Ishigooka T, Yokoyama T., *A Distributed Computing Environment for Embedded Control Systems with Time-Triggered and Event-Triggered Processing*, 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2008, 45–54, http://doi.ieeecomputersociety.org/10.1109/RTCSA.2008.38.
6. Wittenmark B., Nilsson J., Torngren M., *Timing problems in real-time control systems*, ISBN: 0-7803-2445-5, American Control Conference 1995, DOI: 10.1109/ACC.1995.531240.
7. Noriaki A., Takashi S., Kosei K., Tetsuo K, Woo-Keun Y., *RT Middleware: Distrbiuted Component Middleware for RT* (*Robot Technology*), 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems 2005, 10.1109/ACC.1995.531240.
8. Implementation of a Local Exchange Information Mechanism within the Communication Platform for Distributed Control Systems.

Implementation of a Local Exchange Information Mechanism within the Communication Platform for Distributed Control Systems

Abstract: The article shows the way of implementation of a local exchange information mechanism in generic software platform for data exchange, adapted for use in distributed control systems. The paper shows software layers structure responsible for packages transport in scope of single platform instance. Exemplary usage scenarios for local communication mechanisms have been described.

Keywords: communication in distributed control systems, communication platform, data exchange mechanisms

mgr inż. Przemysław Strzelczyk
przemyslawstrzelczyk@gmail.com

Słuchacz studiów doktoranckich na wydziale Elektrotechniki, Automatyki i Informatyki Politechniki Opolskiej – kierunek Automatyka i Robotyka. Pracuje jako specjalista ds. rozwoju oprogramowania w centrum R&D Nokia Networks.

dr hab. inż. Krzysztof Tomczewski, prof. PO
k.tomczewski@po.opole.pl

Politechnika Opolska, Wydział Elektrotechniki, Automatyki i Informatyki, Instytut Układów Elektromechanicznych i Elektroniki Przemysłowej, kierownik Katedry Napędu Elektrycznego, Diagnostyki i Elektroniki Przemysłowej.