POZNAN UNIVERSITY OF TECHNOLOGY ACADEMIC JOURNALS
No 89 Electrical Engineering 2017

DOI 10.21008/j.1897-0737.2017.89.0022

Przemysław STRZELCZYK*
Krzysztof TOMCZEWSKI*

# ADAPTACYJNY SYSTEM WYMIANY INFORMACJI W UKŁADZIE STEROWANIA EGZOSZKIELETU

W artykule przedstawiono koncepcję budowy fragmentu egzoszkieletu ręki. Projekt jest w trakcie realizacji. Dotychczas opracowano system komunikacyjny, pozwalający na tworzenie modułowych oraz rozproszonych systemów sterowania. W systemie zrealizowano mechanizmy komunikacji lokalnej oraz globalnej, pozwalające na wymianę informacji pomiędzy węzłami w obrębie sterowanej jednostki oraz pomiędzy aplikacjami wewnątrz pojedynczego węzła. W systemie zaimplementowano mechanizm automatycznej detekcji modułów.

SŁOWA KLUCZOWE: warstwa pośrednicząca, sterowanie rozproszone, egzoszkielet wspomagany, wspomaganie ruchu

## 1. WSTĘP

W ostatnich latach na świecie odnotowuje się coraz więcej przypadków dotyczących problemów ruchowych, które w znacznej części są powikłaniami po przebytym udarze mózgu. Szacuje się, że jest to jedna z pięciu głównych chorób przewlekłych, mających poważny wpływ na motorykę pacjentów po wypadku [6]. Przy obecnym tempie rozwoju występowanie udaru mózgu oraz chorób pokrewnych ulegnie prawdopodobnie dalszemu wzrostowi [7]. Jednym z częstych powikłań jest częściowy niedowład kończyn. W takich przypadkach pomocny może okazać się fragment aktywnego egzoszkieletu wspomagającego ruch. Może on pełnić zarówno funkcję wspomagania ruchu jak i elementu wspomagającego ćwiczenia rehabilitacyjne. Wskazane jest, aby urządzenie takie miało budowę modułową, pozwalającą na indywidualną konfigurację systemu, zależnie od rodzaju schorzenia.

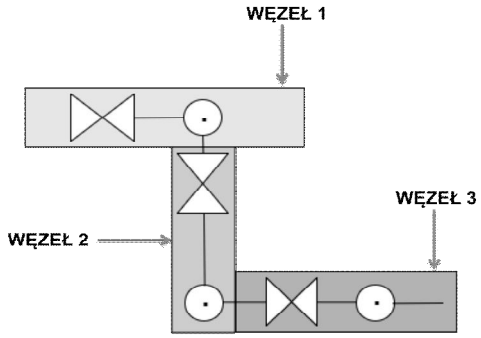
## 2. IMPLEMENTACJA SYSTEMU

### 2.1. Przyjęte rozwiązania sprzętowe

Celem realizowanych badań jest opracowanie fragmentu egzoszkieletu ręki. Projektowany egzoszkielet ma wspomagać ruch człowieka i służyć do celów

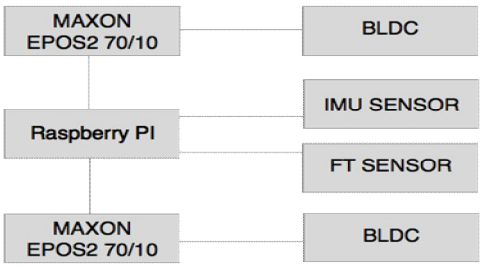
* Politechnika Opolska.

rehabilitacyjnych. Urządzenie będzie bazowało na opracowanym systemie wymiany informacji. System ten umożliwia automatyczną detekcję konfiguracji egzoszkieletu zależnie od tego, które segmenty zostaną włączone w jego skład. Umożliwi to w przyszłości łatwą rozbudowę układu. Na rys. 1 pokazano przyjętą strukturę egzoszkieletu z podziałem na moduły. Realizowany układ zawiera trzy moduły, w skład których wchodzą po dwa złącza kinematyczne. Kolejne węzły systemu odpowiadają za realizację: stawu barkowego, ramienia wraz ze stawem łokciowym oraz przedramienia ze stawem nadgarstkowym. Każdy moduł stanowi autonomiczny element, zawierający napędy wraz z układem sterowania i czujnikami.




Rys. 1. Koncepcja podziału egzoszkieletu ręki na węzły systemu wymiany informacji

W projektowanym układzie moduły egzoszkieletu odpowiadają węzłom opracowanego systemu wymiany informacji. Każdy moduł posiada jednostkę sterującą. Do budowy prototypu systemu wykorzystano moduły Raspberry Pi w wersji 2. Zastosowane moduły posiadają czterordzeniowy procesor ARM Cortex–A7 taktowany z częstotliwością 900 MHz oraz 1 GB pamięci RAM typu LPDDR2. Każdy z modułów zawiera dwa napędy BLDC. Do budowy napędów złącz wybrano sterowniki firmy MAXON EPOS2 70/10 dostosowane do sterowania silnikiem BLDC o mocy do 700 W. Sterowniki te mogą pracować w trzech trybach: prądowym, sterowania prędkością oraz pozycjonowania. W projekcie wykorzystany zostanie tryb pozycjonowania. Do napędów złącz przewidziano silniki BLDC EC90 firmy MAXON o mocy 90 W z wbudowanymi enkoderami typu MILE 800 oraz przekładniami planetarnymi o przełożeniu 66:1. Struktura pojedynczego modułu przedstawiona została na rys. 2. Poszczególne moduły różnią się między sobą liczbą oraz sposobem rozmieszczenia czujników. Do budowy prototypu wybrano czujnik siły nacisku typu rezystancyjnego FlexiForce A201 1lb, posiadający aktywny obszar w postaci okręgu o średnicy 9.53 mm. Czujnik charakteryzuje się zakresem pomiarowym 0–4,4 N oraz czasem odpowiedzi poniżej 5 ms.




Rys. 2. Struktura układu sterowania pojedynczego modułu egzoszkieletu

W chwili obecnej system umożliwia wymianę informacji pomiędzy modułami Raspberry PI. Jego struktura została oparta na robotycznej warstwie pośredniczącej. Aktualnie opracowywany jest prototypu układu mechanicznego oraz realizowane jest oprogramowanie sterujące pracą napędów.

## 2.2. Robotyczna warstwa pośrednicząca

Systemy robotyczne składają się z wielu różnorodnych komponentów, zarówno sprzętowych jak i programowych. Różnorodność stosowanych komponentów ogranicza często możliwości rozwoju projektów, ponieważ większość funkcji programowych jest ściśle powiązana ze sprzętem zastosowanym podczas tworzenia projektu bazowego. Zmiana pojedynczego komponentu może powodować konieczność przeprojektowania architektury całego systemu. Problem ten został zauważony przez szereg firm, szczególnie realizujących duże projekty. Aby oddzielić warstwę logiki programowej od warstwy sprzętowej i umożliwić w ten sposób tworzenie generycznego oprogramowania, opracowano robotyczne warstwy pośredniczące (ang. *Robotic Middleware*). Przykładem instytucji zajmującej się rozwojem warstwy pośredniczącej dla własnych projektów jest m. in. NASA (ang. *National Areonautics and Space Administration*). Agencja rządowa Stanów Zjednoczonych zadecydowała, że wraz ze wzrostem zapotrzebowania na rozwiązania robotyczne, związane z odkrywaniem przestrzeni kosmicznej, koniecznym będzie skonstruowanie systemu pozwalającego na szybkie tworzenie projektów, z zastrzeżeniem ich generyczności oraz modularności [1, 2]. Pojęcie generyczność oznacza możliwość stosowania tego samego oprogramowania sterującego niezależnie od wykorzystywanej platformy sprzętowej. Przykładowymi systemami dostępnymi na rynku, udostępniającymi tego typu funkcjonalności są CLARAty oraz ROS [3]. Są to systemy opracowane do realizacji dużych projektów. Każdy z wymienionych systemów charakteryzuje się inną architekturą, zależnie od przewidywanego obszaru zastosowania. Głównym elementem, na którym bazuje każdy system pośredniczący, jest podsystem komunikacyjny. Podsystem ten powinien udo-

stępniać możliwość przesyłania nielimitowanej ilości informacji pomiędzy modułami systemu rozproszonego, bez potrzeby tworzenia skomplikowanego oprogramowania komunikacyjnego od podstaw. Kolejną ważną cechą systemów pośredniczących jest możliwość dynamicznej rekonfiguracji topologii systemu, bez dokonywania zmian w oprogramowaniu. Projektant lub nawet użytkownik docelowy, mają możliwość dołączenia lub odłączenia dowolnej liczby modułów sterujących do lub z systemu. System tego typu powinien odpowiednio zareagować na zmiany topologii. Podstawową wadą istniejących dużych systemów jest ich przystosowanie głównie do realizacji bardzo dużych projektów oraz powiązanie z wyspecjalizowanymi, pod kątem ich potencjalnego zastosowania, bibliotekami. Powoduje to wydłużenie wstępnych prac nad systemem, ponieważ wymagają czasochłonnej konfiguracji.

Prezentowane w tej pracy rozwiązanie nie wymaga skomplikowanej konfiguracji wstępnej systemu, co umożliwia wykorzystanie go w projektach małych i średniej wielkości. System wymiany informacji przeznaczony został do budowy rozproszonych oraz modułowych systemów sterowania.

## 3. ADAPTACYJNY SYSTEM WYMIANY INFORMACJI

Adaptacyjny system wymiany informacji prezentowany w tej pracy został opracowany w ramach realizowanej pracy doktorskiej. Rozwiązanie jest sukcesywnie rozszerzane o nowe moduły. Obecnie system jest testowany pod względem spełniania założeń architektonicznych. Został on zaprojektowany w taki sposób, aby mógł zostać zintegrowany z istniejącym systemem bez wykonywania poważnych zmian w jego architekturze. System wymiany informacji pozwala na stworzenie zdecentralizowanego, rozproszonego systemu sterowania. Jego architektura pozbawiona jest jednostki centralnej i pozwala na kreowanie systemu mniej podatnego na awarie [8]. W przypadku zcentralizowanego podejścia system zgodnie z regułami podanymi przez projektanta wyłania lidera, który kontroluje jego pracę. Opracowane rozwiązanie nie ogranicza możliwości tworzenia oprogramowania pozwalającego na dynamiczny wybór lidera, który wraz z odpowiednio zaimplementowaną redundancją danych, pozwoliłby na ograniczenie prawdopodobieństwa awarii. Opracowana architektura systemu wymiany informacji składa się z kilku podstawowych elementów:

– **GNCL** (*Global Node Communication Layer*) – warstwa odpowiedzialna za komunikację globalną. Zadaniem tej warstwy jest nawiązywanie połączeń z innymi modułami obsługiwanymi przez system wymiany informacji. Warstwa transportuje informacje na zewnątrz modułu oraz przekazuje informacje skierowane z zewnątrz do aplikacji uruchomionych wewnątrz systemu. Komunikacja odbywa się z wykorzystaniem protokołu TCP/IP [5].

– **LNCL** (*Local Node Communication Layer*) – warstwa odpowiedzialna za komunikację lokalną. Zadaniem tej warstwy jest nawiązanie połączeń lokalnych pomiędzy aplikacjami znajdującymi się w obrębie jednego modułu. Warstwa ta udostępnia możliwość budowania aplikacji opartej o architekturę klient – serwer. Komunikacja odbywa się z wykorzystaniem protokołu TCP/IP [5].
– **NDL** (*Node Discover Layer*) – warstwa odpowiedzialna za detekcję modułów znajdujących się w obrębie tej samej sieci. Zadaniem tej warstwy jest wykrywanie modułów oraz dostarczanie informacji o wersji ich oprogramowania oraz ewentualnych usługach udostępnianych w obrębie wykrytego modułu. Detekcja węzłów odbywa się za pomocą protokołu UDP [5].

W opracowanym systemie podstawową jednostką jest węzeł, który w odróżnieniu od implementacji The Robot Operating System nie jest procesem a modułem. Topologia opracowanego systemu podobnie jak w The Robot Operating System została oparta o strukturę grafów. Określanie modułu węzłem pozwoliło na ograniczenie poziomu skomplikowania topologii systemu. W obrębie jednego fizycznego modułu sprzętowego może działać wiele aplikacji. Wymuszałoby to ich automatyczną rejestrację w systemie wymiany informacji przy starcie kolejnych aplikacji. Może to prowadzić do zbędnego obciążenia systemu wymiany informacji. W opracowanym rozwiązaniu aplikacja uruchomiona w obrębie danego modułu może, ale nie musi, podejmować próby rejestracji w systemie wymiany danych. Może dokonać tego również dopiero w trakcie swojego działania bez żadnych ograniczeń czasowych. Na rysunku 3 przedstawiono fragment kodu realizujący rejestrację programu sterującego w adaptacyjnym systemie wymiany informacji.

```
transportAddress_ =
NaLNCLRegisterTransportAddress(NaSystemGetCurrentProcessPID());
STestLAppRegistrationReq registrationReq;
SLNCLMessage* messageToSend = NaLNCLCreateMessage(transportAddress_,
                                                  TESTL_TRANSPORT_ADDRESS,
                                                  transportAddress_->transportAddr,
                                                  NA_STEST_APP_REGISTRATION_REQ,
                                                  &registrationReq, sizeof(registrationReq),
                                                  MESSAGE_LEVEL_NORMAL);
NaLNCLSendMessage(transportAddress_, messageToSend, messageToSendSize);
```

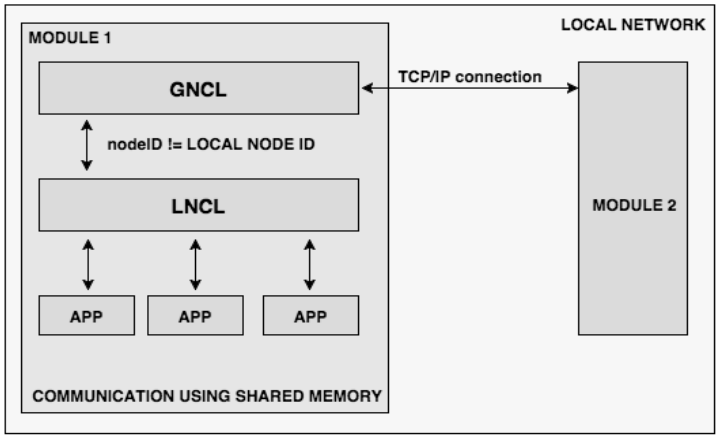
Rys. 3. Przykładowy kod rejestracji w adaptacyjnym systemie wymiany informacji

Tworząc oprogramowanie sterujące napędami można komunikować się z dowolnym modułem znajdującym się w opisywanym systemie, po uprzednim zarejestrowaniu się w warstwie LNCL. Rejestracji dokonuje się za pomocą funkcji API o nazwie *NaLNCLRegisterTransportAddress*. Funkcja po poprawnym wykonaniu zwraca unikalny w obrębie węzła transport adresowy, pozwalający na dwustronną komunikację. Aby utworzyć wiadomość o zdefiniowanym rozmiarze należy skorzystać z funkcji *NaLNCLCreateMessage*. Korzystając z tej funkcji należy podać swój adres transportowy, adres transportowy odbiorcy, identyfikator wiadomości oraz priorytet z jakim wiadomość zostanie przekazana. Finalne wysłanie wiadomości odbywa się za pomocą funkcji *NaLNCLSendMessage*, do której przekazuje się wcześniej utworzoną wiadomość. Na rysunku 4 przedstawiono sposób odbioru wiadomości.

```
SLNCLMessage* receivedMessage =
NaLNCLReceiveMessage(transportAddress_,sizeof(STestLAppRegistrationResp));
```

Rys. 4. Przykładowy kod odbioru wiadomości

W celu odebrania wiadomości w obrębie systemu wykorzystuje się funkcję *NaLNCLReceiveMessage*, podając kolejno swój unikalny adres transportowy oraz przewidywany rozmiar odbieranej wiadomości. Rozmiar przekazywany jako parametr do funkcji jest rozmiarem przekazywanych danych bez nagłówka. Na rysunku 5 przedstawiono schemat przepływu informacji w opracowanym adaptacyjnym systemie wymiany informacji, rozpoczynając od aplikacji, a kończąc na węźle zewnętrznym.




Rys. 5. Przepływ informacji pomiędzy aplikacją a węzłem zewnętrznym [5]

Po wysłaniu wiadomości, której odbiorca nie jest odbiorcą lokalnym (jego pierwsza 16 bitowa część adresu transportowego nie odpowiada adresowi lokalnego węzła), trafia ona kolejno do warstwy obsługującej lokalny routing danych (LNCL), a następnie po wykryciu zewnętrznego adresu transportowego, trafia do globalnego zarządcy wiadomości (GNCL). Globalny zarządca wiadomości sprawdza czy w swojej bazie danych posiada aktywne połączenie z potencjalnym odbiorcą i jeśli tak, to przekazuje wiadomość. Po stronie węzła odbiorczego następuje przepływ informacji w odwrotnym kierunku (od GNCL do aplikacji) [5].

Testowanie opisanego rozwiązania jest zadaniem skomplikowanym, głównie ze względu na rozproszoną budowę oraz możliwość dynamicznej zmiany struktury węzłów. Do testowania prawidłowości realizacji przyjętych założeń przygotowano stanowisko umożliwiające konfigurowanie środowiska składającego się z ośmiu węzłów opracowanej platformy komunikacyjnej. Zastosowaną do przeprowadzenia testów platformą sprzętową było Raspberry Pi w wersji 2. Moduły zostały połączone ze sobą przewodowo za pomocą routera firmy Mikrotik 750G, a następnie zamontowane w opracowanym na potrzeby badań stanowisku pomiarowym. By uniknąć testowania manualnego oraz manualnej analizy uzyskanych danych, utworzono dodatkową warstwę platformy odpowiedzialną za automatyczne wykonywanie zdefiniowanych przez użytkownika testów. Warstwa ta pozwala jednak wyłącznie na przeprowadzenie testów w obrębie lokalnym. Poprawność komunikacji globalnej jest testowana poprzez symulację jednej ze stron połączenia warstwy GNCL. Warstwa TestL (ang. *Test Layer*) udostępnia możliwość uruchomienia specjalnie przygotowanej aplikacji testowej, wykonania dowolnego kodu w ramach jej kontekstu a następnie zebrania i prezentacji wyniku.

W celu sprawdzenia funkcjonowania komunikacji lokalnej realizowanej przez warstwę LNCL przeprowadzono test przesyłania pakietów o stałej wielości pomiędzy dwoma procesami. W tabeli 1 przedstawiono uzyskane czasy przesyłania kolejnych pakietów o rozmiarze 1048 bajtów pomiędzy dwoma aplikacjami. Test ograniczono do przesłania 1024 pakietów o podanym rozmiarze. Wartości zamieszczone w tabeli 1 obejmują tylko część uzyskanych wyników. Średni czas przesłania jednego pakietu w 1024 próbach wynosił 0,02787 ms, przy czym platforma nie została uruchomiona w systemie czasu rzeczywistego.

Dla uzyskania krótszych czasów przesyłania pakietów podjęto decyzję o zmodyfikowaniu w kolejnej wersji oprogramowania priorytetów kluczowych procesów platformy w systemie operacyjnym. Podczas testowania oprogramowania platformy zmierzono również czas przygotowania wszystkich warstw do trybu gotowości. Średni czas przygotowania platformy bez zmiany priorytetu procesów oraz z włączonymi logami debugowymi (ang. *debug logs*) wyniósł 63,32 ms.

Tabela 1. Wyniki testu warstwy LNCL

| Numer pakietu | Czas przesyłu [ms] | Numer pakietu | Czas przesyłu [ms] | Numer pakietu | Czas przesyłu [ms] |
|---|---|---|---|---|---|
| 0 | 0,06992 | 8 | 0,021792 | 16 | 0,000427 |
| 1 | 0,043983 | 9 | 0,021404 | 17 | 0,008648 |
| 2 | 0,023618 | 10 | 0,021388 | 18 | 0,023158 |
| 3 | 0,022417 | 11 | 0,021331 | 19 | 0,021553 |
| 4 | 0,02229 | 12 | 0,021425 | 20 | 0,021509 |
| 5 | 0,021878 | 13 | 0,021412 | 21 | 0,021364 |
| 6 | 0,021489 | 14 | 0,059337 | 22 | 0,021401 |
| 7 | 0,023894 | 15 | 0,057931 | 23 | 0,021306 |

## 4. PODSUMOWANIE

Opracowanie robotycznej warstwy pośredniczącej wymaga dużego nakładu pracy, jednak przy dobrze zaplanowanej architekturze ułatwia ona w kolejnych etapach tworzenie oprogramowania sterującego systemem robotycznym. Taką koncepcję przyjęli m. in. projektanci systemu NASA i innych dużych systemów [9]. Większość tego typu systemów powstawała przez wiele lat, a ich architektura została dostosowana do realizacji bardzo dużych systemów sterowania. Stanowi to duże utrudnienie w przypadku realizacji małych i średnich projektów. Inne systemy zostały przystosowane z kolei do realizacji wąskiej grupy zagadnień, co utrudnia ich wykorzystanie w innych zastosowaniach.

Rozwiązanie zaproponowane w tej pracy ma na celu opracowanie systemu o dużej uniwersalności, nie wymagającego długotrwałej konfiguracji. Zostało ono przystosowane do realizacji systemów o różnej wielkości. Jest uniwersalne, ponieważ nie posiada wbudowanych modułów sterujących konkretnymi urządzeniami wykonawczymi. Cechuje je natomiast generyczność. Projektant systemu ma do dyspozycji platformę łączącą wszystkie elementy systemu, umożliwiającą automatyczną konfigurację systemu, rejestrowanie nowych i wyrejestrowywanie usuwanych węzłów. Dzięki temu system może być wykorzystywany w układach o zmiennej konfiguracji lub w układach o strukturze modułowej. System ten posiada mechanizmy umożliwiające jego autokonfigurację, tzn. automatycznie rozpoznaje, jakie moduły zostały podłączone i umożliwia wymianę informacji pomiędzy nimi. Mechanizmy te zostały rozbudowywane o reguły reakcji na zdarzenia zewnętrzne oraz wytworzenie odpowiedniej abstrakcji sprzętowej (ang. *Hardware Abstraction Layer*), pozwalającej na swobodny wybór urządzeń pomiędzy dostępnymi wariantami.

Opracowany system nie posiada natomiast wbudowanych bibliotek udostępniających funkcjonalności związane ściśle z zagadnieniami sterowania urządze-

niami wykonawczymi. Udostępnia jednak wtyczki dla aplikacji sterujących. Powoduje to, że system jest uniwersalny i może być wykorzystywany do realizacji szerokiej gamy zagadnień.

## LITERATURA

[1] R. Volpe, et.al. "The CLARAty architecture for robotic autonomy," Proc. of IEEE Aerospace Conf., Montana, Marzec 2001.
[2] I. A.D Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, NASA Ames Research Center, Moffet Field, Sunnyvale, Marzec 2003.
[3] S. Ceriani, M. Migliavacca, „Middleware in robotics", Internal Report For „Advanced Methods of Information Technology for Authonomous Robotics",Politecnico di Milano.
[4] R. Volpe, I. A.D. Nesnas, T. Estilin, D. Mutz, R. Petras, H. Das, „CLARAty: Coupled Layer Architecture for Robotic Autonomy", California Institute of Technology, Grudzień 2000.
[5] Strzelczyk P., Tomczewski K., Realizacja mechanizmu lokalnej wymiany informacji w ramach platformy komunikacyjnej w rozproszonych systemach sterowania, „Pomiary Automatyka Robotyka", R. 20, Nr 1/2016, 65–68, DOI: 10.14313/PAR_219/65.
[6] The World Health Report 2008, http://www.who.int/whr/2008/en.
[7] J. Huang, W. Huo, W. X, S.Mohammed, Y. Amirat, „Control of Upper–Limb Power–Assist Exoskeleton Using A Human–Robot Interface Based on Motion Intention Recognition", IEEE Transactions on automation science and engineering, DOI: 10.1109/TASE.2015.2466634
[8] Schlegel Ch, „Communication Patters as KeyTowards Component–Based Robotics", Advanced Robotic System International, University of Applied Sciences Ulm.
[9] Schegel Ch, Steck A. Brugali D., Knoll A. „Design Abstration and Processes in Robotics: From Code–Driven to Model–Driven Engineering", University of Applied Sciences Ulm.

## ADAPTIVE EXCHANGE INFORMATION SYSTEM DEDICATED FOR THE CONTROL SYSTEM OF EXOSKELETON

The article presents the concept of upper limb exoskeleton construction. The project is in progress. The communication system which allows for the creation of modular distributed control systems is developed. The system consists of local and global communication mechanisms allowing for the exchange of information between nodes within the controlled entity and between application in scope of a single node. The system implements a mechanism for automatic detection of modules.

*(Received: 12. 02. 2017, revised: 28. 02. 2017)*