Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2009) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

Platform for distributed execution of agents for trusted data collection

Emil Gatial[a], Zoltán Balogh[a], Ladislav Hluchý[a] *

[a]*Institute of Informatics, Slovak Academy of Sciences, Bratislava, 84507, Slovak Republic*

Abstract

In this article we aim at achieving trusted data collection from heterogeneous distributed sources including legacy systems and human end users. We adopt an agent-like approach where we are able to send agents to host platforms connected to legacy systems. Functionality of agents may vary while we are solving the trust between host platforms, legacy systems and agents. We had not used any existing multi-agent system but we extended an existing production grade service platform called Jini and implemented agents above Jini services. The core of this work is about how we adopted the Jini platform to host agents. We also provide examples about the implementation and integration of the platform in a real world example – a crisis management domain. We present our current achievements and plans concerning the implementation of the proposed platform.

Keywords: trust; security; agent platform; remote code execution; process management

1. Introduction

One of the challenging demands of the nowadays distributed information systems is to collect information from legacy systems of various organizations and from human users in order to semi-automatically manage a business process or enact decisions on various management levels. This collection of information should be enacted in a secure manner through IP-based network while ensuring trust between both parties – information consumers as well as information providers. In a typical distributed application setting many actors participate in the process where the competences between all parties are explicitly defined. The gathering of information can be enacted either from legacy systems or from human end-users through electronic devices (computer, mobile device or handheld). Herein we present the design and implementation of a distributed system which fulfills all the above set goals.

In the next chapter we set our main goal and define related objectives for the system we were trying to design. We found concepts of agents and multi agent systems (MAS) as the most appropriate technology to be used in order to fulfill our objectives. Unfortunately none of the available MAS was suitable for production-grade implementation of agents. We found out that a framework called Jini [1] has many of the features (some are described in [2]) we were looking for. Unfortunately Jini framework is more service-oriented than MAS. Therefore we had to extend the Jini and implement an agent-like platform above Jini services. The third chapter suggests identified subsystems and core agents needed in our platform. The fourth chapter briefly introduces the Jini framework and presents how we

* Corresponding author. Tel.: +421-2-59411-308 ; fax: +421-2-5477-1004 .
E-mail address: emil.gatial@savba.sk .

had used Jini services to send and host agents. Examples of three core agents (most frequently used) are also introduced – an agent which delivers information from legacy information systems (IS), an agent for communicating with users and an agent which is able to configure IP devices. The viability of our platform is validated in a real crisis management infrastructure which is described in the fifth chapter. The last chapter concludes the article as well as presents our current achievements and plans concerning the implementation of the proposed platform.

2. Objectives, challenges and requirements for the platform

Our main goal is to design and implement a platform for distributed execution of agents for trusted data collection from distributed legacy systems and users where multiple actors are involved from various organizations with different competences and communicating over IP-based networks. In such settings the requirements exist for secure communication and trusted collection of data from various sources. The role of agents in such architecture is primarily coordinated collection of information. The gathering of information is enacted either from legacy systems or from human end-users through mobile devices by guided dialog. In respect to requirements the overall agent infrastructure must be a secure, robust and fail resistant system.

2.1. Platform challenges

Agent technology was selected due to the ability to fulfill such requirements through support of mobile and dynamically deployable executable code. We have decided to use agents for our platform in order to solve the following challenges:

1. Obeying network failures – in case of temporal network failure, the agent code is still running and the results are sent when the connection to the network is back even for a small time period;
2. Protecting sensitive data – management of centrally stored sensitive data is more convenient way for their protection presuming that such data are defended by cryptography facilities;
3. Sensing remote events and monitoring – agents can be deployed on remote systems to track and monitor events;
4. Handling unexpected events – agent code should not be able coping with exceptional events occurring during its execution. In such cases, the centrally managed agent system is able to react in more convenient way by issuing the other agents solving an exceptional state;
5. Uploading code on demand – a way to upload a code with certain functionality to the agent's host environment;
6. Trust and security – trust must be established between the agent, host platform and any other systems communicating with the agent while communication between these entities must be secured.

The most critical challenges among all the above mentioned ones are security and trust. According to [3] the platform from which an agent originates is referred to as the home platform, and normally is the most trusted environment for an agent. Agents are mainly executed on remote sites which provide the computational environment in which agents operate. We will refer to these sites as the host platforms (or agent platforms). There are certain security and trust risks which must be addressed between the host platform, agent and other systems (more details are discussed in [3]):

- Host platform attacking the agent: The main requirements from the agent-side are laid out in respect to the "malicious host problem" [4] requiring isolated execution environment for agent execution preferring dedicated isolated hardware; means to attest the platform required in order to detect if the host platform is in trusted state.
- The agent attacking the host platform: Reversely, a host platform has the following requirements in respect to agents: the agents must be executed in isolated environment, so they can not harm legacy systems. In order to track agents, any agent in the platform must be cryptographically signed. Only agents signed with trusted authority and assigned to selected category will be trusted by a host system.
- The agent attacking another agent: It is required that any agent which will be used in the system will need to be audited and certified by a central authority. In turn, every host platform will be configured to execute only certified agents. These two security policies should ensure that malicious agents will not be deployed into the infrastructure. Only a breach of the set security policies might lead to potential agent-to-agent security risk.
- Other entities attacking the agent system: Agents will also connect to the legacy systems (third party software). Therefore there exists a risk of an agent being attacked by a legacy system but also vice versa the risk of attacking legacy system by an agent. Another requirement arises from agents to the host platform is secure

protected connection to legacy systems. Physical security of network connection can be achieved either by direct cable connection of the host platform with legacy system or by managed network security.

2.2. Agents and multi-agent systems

The concept of an agent provides a convenient and powerful way to describe a complex software entity that is capable of acting with a certain degree of autonomy in order to accomplish tasks on behalf of its user. Unlike objects, which are defined in terms of methods and attributes, an agent is defined in terms of its behavior [5]. The definition for an agent according to the "Agent's Bible" by Wooldridge and Jennings [6] is the following: "An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives." Depending on how an agent behaves many types of agents including reactive agents, knowledge-based agents, goal-based agents, problem solving agents, practical reasoning agents, utility based agent, intelligent agents or mobile agents were defined.

In [8] Wooldridge and Jennings define a Multi-Agent System (MAS) as follows: "A set of agents that interacts in a common environment to achieve common or individual goals is called a multi-agent system (MAS)". Agents in a MAS are typically capable of co-operating to solve problems that are beyond the abilities of any individual member. In a MAS agents attempt to satisfy collaborative global goals as well as their own local goals. MAS are often heterogeneous, i.e., composed of different kinds of agents having different roles, where the agents can share knowledge using any agreed language, for example using Knowledge Query Manipulation Language (KQML) or FIPA's Agent Communication Language (ACL). Moreover, in MAS the goal can be shared among the agents or alternatively the goal of agents can be managed by central designer system [7]. The agents in MAS have several important characteristics [8]: *Autonomy:* the agents are at least partially autonomous or fully autonomous; *Local views:* no agent has a full global view of the system, or the system is too complex for an agent to make practical use of such knowledge; and *Decentralization:* there is no designated controlling agent.

2.3. Requirements for the platform

Despite we had reviewed many agent platforms including Jade, Retsina, Aglets, Grasshopper and others, neither of them was suitable for production grade implementation of agents. Therefore we decided to look for a distributed platform already used in production systems which provides as required features as possible. The main of many requirements for our framework should follow the possibilities for a simple but powerful discovery mechanism, solved remote communication between entities, easy implementation of trust and security mechanisms, introducing as many distributed entities as required on demand, implementing mobile code and distributed management.

Our main priority is therefore not to implement a fully FIPA-compliant agent but rather to fulfill our specific above stated requirements as well as possible. Unlike in MAS where the agents can manifest self-organization and complex behaviors, in our approach the organization of agents relies mostly on the decision made by semi-centralized process management system. In our approach we do not implement any specific agent communication language but rather use the communication infrastructure of the used platform. If there exists some centralized coordination system the agents' behavior might be viewed as merely sensors and effectors for executing environment. Anyhow we design our platform as centralized because any business process management requires centralized control. In our settings a problem needs to be solved in a distributed fashion either because a central controller is not permanently available or because one wants to avoid compromising or issue of resources.

2.4. Jini – framework of our choice

The homepage of Jini states the following: "Jini technology is a service oriented architecture that defines a programming model which both exploits and extends Java technology to enable the construction of secure, distributed systems consisting of federations of well-behaved network services and clients. Jini technology can be used to build adaptive network systems that are scalable, evolvable and flexible as typically required in dynamic computing environments." Because of these statements, which more or less match our requirements, we have decided to adopt Jini platform to host our agents for distributed and trusted data collection.

In Fig. 1 we present the basic entities and basic model of Jini technology. In a running Jini system, there are three main players. There is a Service (Service Provider or Server), Client and a Lookup Service. All three components are connected by an IP based network. Code can be moved around between these three components. Service can be an implementation of any service. Client is generally interested in invoking the Service. Service can announce itself through uploading a special code called ServiceProxy (SP) to the Lookup Service. ServiceProxy is simply a proxy gateway to the original Service. A Client can download the ServiceProxy and through its invocation make use of the Service. Registrar is used as a reference object for both the Service and the Client.

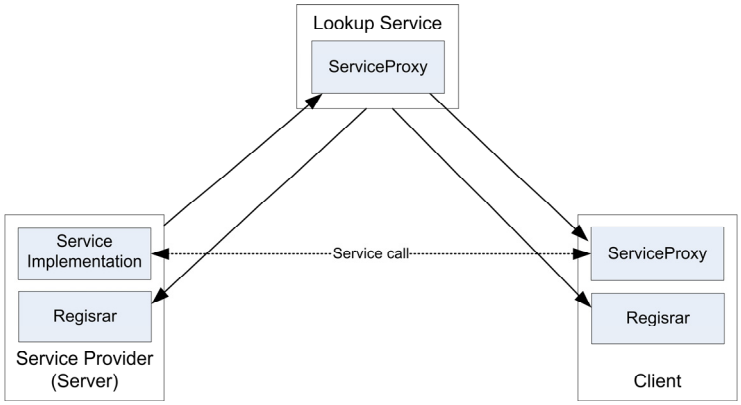

Fig. 1. The concept of Jini service invocation

There are several things which can happen between the Client, the Service and the LookupService (LS):

1. Discovery – Client discovers SP of a Service through LS either by unicast request, in case it knows the actual location of the LS, or by multicast discovery;
2. Join - Once the LS is found it returns a Service Registrar (SR) object to the service to register the service in the LS. This includes the unique service ID, the proxy object of service implementation and service attributes;
3. Searching for services - It is done by consulting the lookup catalog on the LS and matching suitable parameters of a service based on the type, name or service attributes. The LS will release a Java proxy object, which specifies how to connect service directly. This feature makes Jini more powerful than classic RMI that requires the service to know the location of the service deployment in advance;
4. Leasing - Since Jini services will not necessarily be permanently available, the concept of leasing is introduced. When a service registers into some LS, a lease is granted for the certain time duration (or by default the lease is granted forever). If the service lease is granted for exact time duration, the lease needs to be periodically renewed in order to check a service is still alive.
5. Object serialization - Jini serializes the Java objects in order to be sent across the network. This means that Java object can be accessed remotely as if it were local, as opposed to specifying service access protocols like in service oriented architectures (SOA).

Using Jini framework usualy brings many improvements in comparison with SOA, but it suffers from some inconveniencies; i.e., the Jini event model does not guarantee the order of events occurring across a network, therefore the event messages are usually indexed.

3. Subsystems and core agents

In order to fulfill the architectural requirements set the infrastructure was decomposed into subsystems. These subsystems are related and will cooperate together using defined interfaces. The list of subsystems is given in the Table 1.

Table 1. Subsystems of the architecture

Agent	Functionality
Information Delivery Agents (IDA)	IDA agents will need to connect to legacy information systems of third parties to retrieve information about available resource capacities
User Communication Agent (UCA)	It communicates with users in a form of guided dialog through electronic device. Will include authentication and interface to authorization of the user.
IP Agent (IPA)	An agent able to configure IP devices such as routers.

The purpose of the Distributed Secure Agent Platform (DSAP) is to provide an execution environment for different types of agents. The main aim of Process Management Subsystem (PMS) is execution of processes and coordination of involved agents in the emerged crisis situation. The plan scenario for each type of crisis situation will have to be pre-prepared in form of an abstract process. The exact execution of such plan in a concrete situation will depend on the context of the crisis situation. The agents available within the proposed infrastructure have to be stored on Trusted Servers, from which they can be requested for deployment on the side of HPs – this functionality is encompassed within the Agent Repository (AR). Public Key Infrastructure (PKI) will allow certifying, and subsequently verifying, all the objects deployed in the infrastructure. Agents will require information about the information sources which can be queried in order to retrieve information about resource availabilities. Resource Inquire System (RIS) will provide an interface which will provide such capability. Additionally there is a set of core agents which are required in order to ensure functionalities of the architecture. List of agents including their brief functionality description is in Table 2.

Table 2. Core agents used in the architecture

Subsystem	Basic description and functionality
Distributed Secure Agent Platform (DSAP)	The core agent platform. It provides means for agent deployment, execution, migration and communication.
Process Management Subsystem (PMS)	Based on the plan collected from users. It will generate and execute a plan of activities.
Agent Repository (AR)	Database of system users, agents and their certificates. Process of accreditation of agents.
Public Key Infrastructure (PKI)	Certification and verification of agents, users and resources.
Resource Inquire System (RIS)	Giving the information which system to query for specific information.

4. Extending JINI to host agents

The following paragraph explains the agent code migration and agent lifecycle, while the next one introduces the security mechanisms of this agent platform.

Since distributed algorithms were elaborated by computer science, the designed platform focuses merely on remote code execution with emphasis on establishing trust among multiple parties. Moreover, the agents in our scenario are designed to access resources in order to help a person to make a decision but not to make a decision autonomously. This does not express that the agents are meant to execute only simple code accessing resources (i.e. querying DB systems or temperature sensors). The selection of resources or resource providers can be optimized within the agent's code, while a responsible person has a full control upon agent's activities. Such optimizations of resource selection are beyond the focus of the agents described in this paper. According to [9] the platform from which an agent originates is referred to as the home platform and the platform where an agent's execution takes place is called host platform (HP). In general, any party which wishes to join the implemented architecture and to provide information from their legacy systems or users must introduce a host platform for agents. We refer to such parties as host platform providers. One of the main focuses of designed agent platform is on securing and migration the agent from a home platform to a HP and on their mutual communication.

DSAP is built upon the Java and Jini framework making use of the service proxy's remote method call feature for modeling a HP. The HP, issuing an agent, locates possible service providers by querying LS(s) (either using known LS location or finding all possible LS by multicast) for specific DSAP service proxy. The DSAP services registered in LS usually hold additional service attributes (service metadata) like service location and service capabilities. The DSAP client discovers the right DSAP service implementation by matching the required service attributes with DSAP service attributes stored at LS and makes use DSAP proxy instance to access the HP.

The lifecycle of agent emerges the following issues:

- Finding suitable DSAP service in order to deploy an agent with specific goal. In order to successfully accomplish the goal of an agent, the suitable DSAP service has to be identified. One way how to successfully identify the suitable DSAP service deployment is to discover every DSAP service among LS and filter out the most appropriate service by service attributes' matchmaking. This process includes comparison of DSAP service attributes with goal specific attributes of an agent. Such process should be done every time the agent has to be deployed, which leads to cumbersome and time consuming operations. Another way is to implement special kind of service whose special goal is to register event listener notifying a service attribute change among discovered available DSAP services. This kind of service should periodically discover new DSAP services or discard no longer available services. RIS service should implement methods searching for the most appropriate DSAP service according to specific criteria, for example to find the DSAP service deployed in the vicinity to given geographical location, etc.
- Deployment of an agent in short- or long-term manner using DSAP client. The short-term agent deployment occurs in on-demand information acquisition where an agent responds almost immediately to a DSAP client with return messages. Typically, the agent is terminated and cleared from HP or suspended to be used later on. In case of the long-term deployment, the agent may reside on the HP and respond to a DSAP client continuously or may check the HP environment by invoking events on DSAP client.

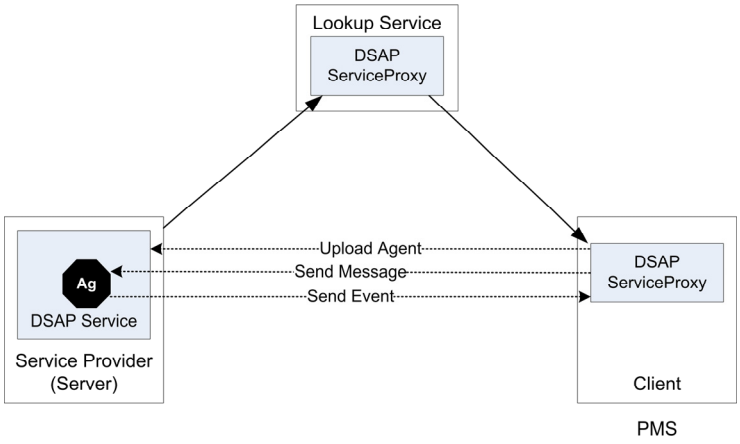

Fig. 2. The concept of DSAP based on the Jini framework

In Fig. 2., the process of discovery, join and agent communication is shown within the scope of the DSAP services. Here, HP discovers LS and joins the registrar object. When a DSAP client (residing in some home environment) wants to deploy an agent it searches the LS for the DSAP services conforming the attributes of an agent. Commonly, the attributes describe a location of a DSAP service, the capabilities reflecting the actions that can be made by an agent using specific Java libraries and an organizational unit that an agent needs to cooperate with. This attribute concept can be easily extended using other attribute types. Once an appropriate DSAP service was found an agent is uploaded and run in HP environment under the control of the DSAP service. Each agent is assigned with globally unique identifier (GUID) before the process of deployment will take place in order a client can communicate with specific agent. A DSAP client is able to send messages to agent using GUID and receive immediate response, or a DSAP client is notified by firing the events processed by event handler of DSAP client.

Security mechanism for agent platform is incorporated into the DSAP service using PKI standards with respect to security requirements on the agent platform. Moreover, the DSAP relies on a Secure Docking Module (SDM)

storage holding private keys and a Trusted Docking Station (TDS) quoting a trusted platform state. The HP, deployed within the TDS, is measured by auditing the BIOS and operating system booting sequence measurements evaluated as SHA-1 hash values and stored in a Trusted Platform Module (TPM). The SDM only releases private keys if the host platform adheres to a configuration (trusted state) that enforces a key protection policy. The root of trust is established between the agents' home platform and HP by audited agent code before its usage will take place. The audit process must ensure that the agent does only what its creator states it should do, and that it does not contain any malicious code, which may jeopardize the integrity of the HP. Establishing the trust between an agent and a HP is depicted in Fig. 3.

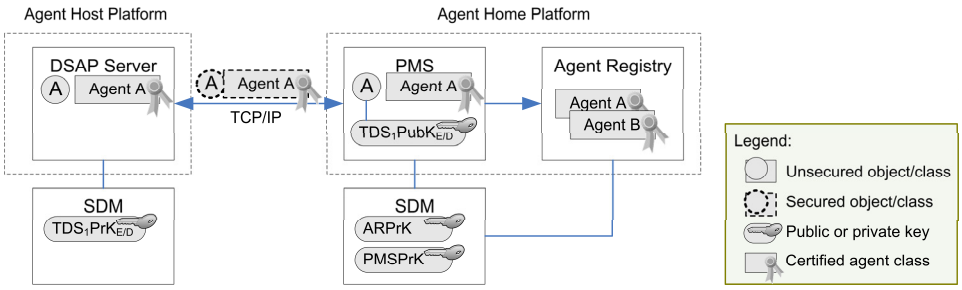

Fig. 3. The scheme of DSAP concept to establish secure and trusted communication of agents

Agent repository (AR) holds the set of certified agent Java classes or jar files. The code of agents may vary from executing simple DB query to complex management of HP resources. It is up to agent designer to implement an agent's functionality, but with respect to the fact that the code must be audited and certified whether by the HP provider or by trusted third-party authority. Based on the code certification the HP provider can trust the code running his or her HP. When PMS decides to issue an agent, it queries AR to obtain the classes implementing the agent. Here, PMS is able to verify the certificate of agent classes. Next, an instance of agent object is created by PMS where the agent attributes are set. The agent object and its classes are encrypted using AES key secured by $TDS_1PubK_{E/D}$ public key (referred to as key wrapping) of HP. After the encrypted agent is moved on the HP, the DSAP service decrypts the AES key using $TDS_1PrK_{E/D}$ private key of the HP (received from SDM) and uses this key to decrypt an agent. The HPs usually provides access to some resources that a specific agent is able to process. Here, PMS is responsible for choosing the right type of agent and for setting him up to provide the required results. The results are encrypted using the same AES key and sent back to PMS.

5. Basic platform design and core agents

The architecture described herein is being used also within an EU integrated project called Secricom [10]. The implementation of the architecture in the project is called Secure Agent Infrastructure (SAI). SAI addresses timely delivery of relevant information, obtains the information about available resources (material or human) and helps the authorities manage the distribution of the resources. SAI communicates with legacy information systems operated by agencies and institutions involved in crisis resolution. There are several systems to which SAI gets connected. We describe the integration with SDM, PTT – Push-To-Talk system and MBR – Multi Barrier Router systems.

In order to overcome the risks described in section 2.1, agents require safe secured place to store cryptographic credentials (PKI secret keys) and provide interfaces to retrieve these keys, ways to attested platform (execute on a host platform which is in a trusted state) and provide interface to safely communicate with legacy systems. All these functionalities are provided by a SDM hardware module [11, 12]: SDM is a key storage device with local attestation and verification capabilities. SDM establishes trust on the host platform where agents are being executed – called TDS. A trusted state is a specific software configuration. This software configuration is measured by using a TPM. A TPM is a special security chip which provides amongst other functionalities the protected capability of measuring the software configuration of its host device. A TPM must be present in the TDS. The combination of a SDM and a TDS is called a Secure Docking Station (SDS).

SAI uses SDS deployed in a physical proximity of the legacy information system, preferably in the same room and acts as a secured and trusted extension of the Secricom infrastructure. SAI executed in SDS eliminates the

exposure of the legacy IS to the outside world and allows the operator of the legacy IS to have increased trust in the information consuming party. SAI can process the information received from the legacy IS while conserving the network bandwidth, limiting possible exposure of sensitive data – sending back the results only and continuing data processing even if the connection to the outside world is intermittent. More information about these technologies can be found in [11, 12].

Secricom PTT is a client-server voice and data communication system using IP protocol and is developed by Slovak company Ardaco [13]. PTT optimizes and protects the communication of teams without being concerned about misuse of information. Regardless of communication endpoint (mobile, laptop or handheld) the communication is secure and safe. The communication is based on the SAI connecting to PTT servers in order to communicate with users. Concretely UCA agent is being integrated with PTT through using XForm [14] standard for user-to-machine interaction. PTT takes care of delivering and displaying the forms on the user side, while UCA is responsible for the form processing. Forms are being automatically generated by PMS during run-time in accordance to overall process status and process configuration. Integration of UCA with PTT adds a more flexible way of data collection and user communication to Secricom infrastructure (see the scheme in Fig. 4.).

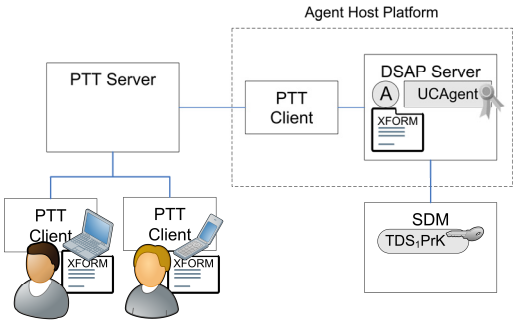

Fig. 4. UCA and PTT communication schema

The Secricom Multi Bearer Router (MBR) is a modular router development platform and is developed by a UK company QinetiQ [15] that delivers the IPv6 network enabling overlay. It provides seamless, ad-hoc end-to-end connectivity and emerging next generation, static and mobile bearers, networks and user devices. SAI integrates with MBR using the IPA agent. MBR must be equipped with SDS in order to provide trusted and attested execution environment for agents. IPA agent is able to configure different properties of network such as communication prioritization or bandwidth control between different bearers.

6. Process management subsystem (PMS)

Since DSAP platform provide convenient mechanism for secured agent migration within trusted environment, the coordination of agents is managed by centralized PMS capable of tracking sequence of actions to achieve specific goal. PMS holds ontological description of processes that can possibly occur when some external event is fired.

Formally, the process P means to execute any possible actions to achieve process goal. The process state or context C denotes set of resources available during a process execution. The process goal G_P is to execute every ending action in the specific process P. Actions templates of the process $AT_P = \{at_{P1}, at_{P2},\ldots, at_{Pm}\}$ hold the set of action templates available for execution in the given process P. Action templates prescribe what action will be instantiated while fulfilling the preconditions of AT_P. Such precondition set $PC_{AT} = \{pc_{AT1}, pc_{AT2},\ldots, pc_{ATn}\}$ specifying the set of resources that must be available before the action will be executed. Here, each AT_P holds references to role of responsible actor R_{AT}, which is mapped to specific actor (person or legacy resource responsible for process invocation) later in the deriving process of action instance. AT_P also describes the type of its effects $E_{AT} = \{e_{AT1}, e_{AT2},\ldots, e_{ATp}\}$ containing the set of activity output types. Each action derived from AT_P produces resources of E_{AT} (subset of Resource ontological concept) that are stored in C. Agents $AG = \{ag_1, ag_2,\ldots, ag_q\}$ contain set of agents that are capable of solving a specific task. Action $A_P = \{a_{P1}, a_{P2},\ldots, a_{Pr}\}$ represents an instance of AT_P coupled with ag_i zero or more agents while resources PC_{AT} are available in C. Formally, the task of PMS is to initiate some process P and to manage execution of actions A_P prescribed as AT_i in order to achieve the goal G_P.

Initially, PMS searches for initial action templates in AT_P, where the precondition is not specified, thus $PC_{AT} = \{\}$, for each process P and therefore instances of such AT_P can be executed without any required resources. For every at_{Pi} where $PC_{AT} = \{\}$ the starting action a_{Pi} is created. The instances of agents ag_j, described as E_{AT} in a_{Pi} are created and deployed in DSAP service in order to communicate with the specific actor matching the role R_{AT} for process P. Actually, the PMS will issue the starting agents which contact actors (persons or legacy resources) to initiate a process P. In case of communicating with a person, the UCA long-term agent is utilized to query the responsible person for initiating the process and in case of communicating with legacy system, the ICA long-term agent is sent to HP to check availability of specific resource. Starting agents usually sit in the HP till the required resource is made available to initiate process P and update the context C. Since multiple processes can be enacted simultaneously, the starting agent is redeployed right after the agent has started a new process.

If specific process P is initiated, PMS searches every action template in AT_P set, where the precondition set PC_{AT} contains a subset of required resources included in context C, formally expressed as selecting AT_P; $PC_{AT} \subseteq C$. For every found action template at_{Pi} the action a_{Pi} is created. The instances of agents ag_j, described as E_{AT} in a_{Pi} are created and deployed in DSAP service in order to communicate with the specific actor matching R_{AT} for process P. Here, agents can be issued as long-term usually for monitoring purposes or short-term requesting immediate response. Each agent a_i produces resources of type e_{ATi} that are included into context C. Each time the PMS changes the context C, a new set of AT_P is found according to selecting AT_P; $PC_{AT} \subseteq C$. The PMS finishes the process execution P if G_P condition is matched.

7. Use case scenario

Herein we present a sample scenario in which coordinated information collection using agents takes place.

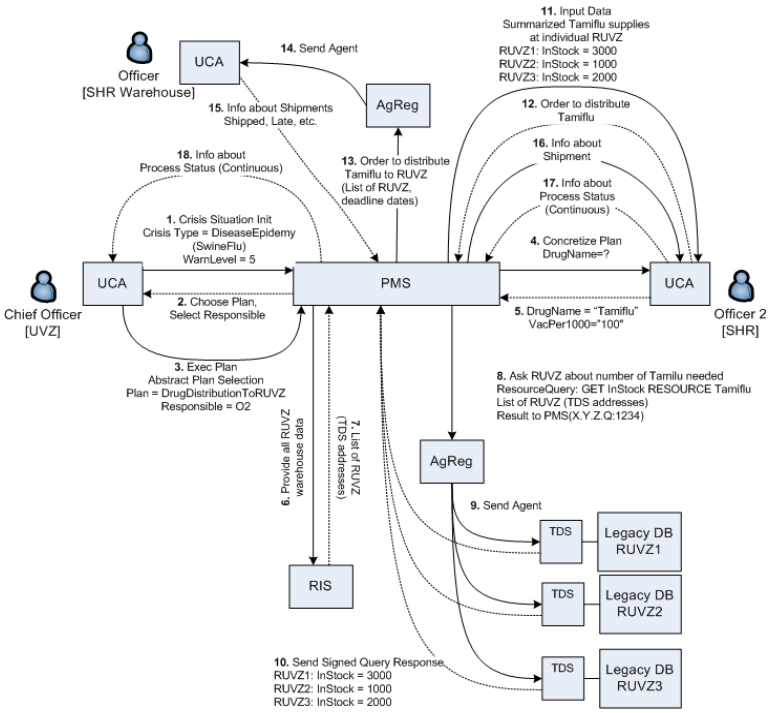

Fig. 5. Scheme of a sample crisis scenario

The scheme in Fig. 5 depicts an imaginary epidemic crisis scenario: A country has a sudden rise in the number of people sick from an epidemic flu. There are many infected people and others are suspected to be sick soon. The organization responsible for mitigation of epidemic is UVZ. Personally a Chief Officer (CO) at UVZ is responsible for such situations. The CO decides to set the warning level to 5. As part of this warning level, UVZ needs to make sure that there are sufficient supplies of vaccines in regional UVZ branches (RUVZ). Such information must be retrieved from legacy systems of each RUVZ. The CO must delegate this information collection to an officer at

another organization called SHR. After the officer at SHR finds out the supplies at individual RUVZ he/she needs to delegate the task for distributing additional amount of vaccines to an Officer at SHR Warehouse. Information about complement shipments of vaccines to RUVZ is sent by SHR Warehouse Officer directly to SHR Officer who redirects this information to the CO at UVZ. For detailed description of the presented scenario refer to [16].

8. Conclusion and future work

In the stage of writing this paper the task responsible for design and development of distributed secure agent platform is being finished. We provided the first prototype in January 2010 integrating DSAP platform with AR, MBR, PMS, SDM and PTT components. The described architecture was demonstrated on the use case called "Swine-flue crisis management" mediating scenario for drug distribution during the pandemic process mitigation. The integration with monitoring center is planned using the Java logging mechanism, where the messages about the state changes within the DSAP environment are sent to the monitoring center and analyzed. In the future we plan to identify other suitable problem domains and to customize the platform for new challenging application domains.

In this article we have analyzed the requirements for agent-based systems and proposed a distributed architecture. We have described the mechanism for selecting and enacting agents deployed in DSAP environment to achieve specific goals. We have decomposed our agent architecture to subsystems and identified several core agents to be used in an architecture implementation. A sample scenario was described, which demonstrates the possible use of individual agents in case of a crisis in a distributed IP-based communication infrastructure. Lastly we sketched our achievements using the proposed architecture within an EU integrated project called Secricom and described future integration and implementation plans. We believe that beside crisis management there are many other application domains where trusted code execution using agents is appropriate to use and where the proposed distributed agent-based architecture would suit as well.

Acknowledgement

This work is supported by projects SECRICOM FP7-218123, APVV DO7RP-0007-08, SEMCO-WS APVV-0391-06, VEGA No. 2/0211/09, VEGA 2/0184/10.

References

1. Jini. URL: http://www.jini.org/wiki/Main_Page

2. G. Aschemann, et al., A Framework for the Integration of Legacy Devices into a Jini Management Federation, In Proceedings of Tenth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'99), 1999

3. M. Wooldridge, An Introduction to MultiAgent Systems, John Wiley & Sons Ltd, 2002

4. N. Borselius, Mobile agent security, Electronics & Communication Engineering Journal, , Volume 14, no 5, IEE, pp 211-218, 2002

5. D. L. Martin, et al., The Open Agent Architecture: A Framework for Building Software Systems Applied Artificial Intelligence, 1999.

6. M. Wooldridge , N. R. Jennings, Intelligent Agents: Theory and Practice, Knowledge Engineering review, 1995

7. Y. Shoham, et al.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations, Cambridge University Press, 2008

8. L. Panait, S. Luke: Cooperative Multi-Agent Learning: The State of the Art. Autonomous Agents and Multi-Agent Systems, 2005

9.W. Jansen, T. Karygiannis: Mobile Agent Security – NIST Special Publication 800-19. National Institute of Standards and Technology, Computer Security Division, Gaithersburg, MD, 1999

10. Secricom project, URL: http://www.secricom.eu/

11. B. Šimo et al.: Security requirements and specification for docking station module. Deliverble report D4.1, the SECRICOM project, 2009

12. D. Hein et al.: Functional specification of the Secure Docking Module. Deliverable report D5.1, the SECRICOM project, 2009

13. Ardaco company, URL: http://www.ardaco.com/

14. The Forms Working Group, http://www.w3.org/MarkUp/Forms/

15. QuinetiQ company, URL: http://www.qinetiq.com/global.html

16. L. Hluchý, Z. Balogh, E. Gatial, Distributed Agent-based Architecture for Management of Crisis Situations using Trusted Code Execution, in Internationnal Synposium on Applied Machine Intelligence and Informatics, 2010