Fault Tolerant Automated Task Execution in a Multi-robot System

Stanislaw Ambroszkiewicz, Waldemar Bartyna, Kamil Skarzynski and Marcin Stepniak

Abstract In multi-robot systems unexpected situations occur frequently, and cause failures of robots performing tasks. Mechanisms for automation of failure handling and recovery (if possible) are proposed. They are based on general protocols similar to the well known standard WS-TX for business transactions. The protocols and mechanisms are implemented in the prototype Autero system as a software platform for accomplishing complex tasks in open and heterogeneous multi-robot systems.

1 Introduction

Robots provide services that can be used to accomplish complex tasks in everyday life, see for example [1].

The Autero system presented in this paper is a software platform for delegating tasks (by human users) to be accomplished in a multi-robot system. The platform is responsible for planning, distributing subtasks to the available robots, monitoring and controlling the subtasks performance as well as handling recovery from failures. All this is done in an automatic way on the basis of generic communication protocols, so that the architecture of Autero is appropriate for open distributed systems consisting of heterogeneous robots providing services.

Since some ideas and methods are adopted form electronic business transactions, realization of a task is called transaction. The transaction is successfully completed, if the delegated task is accomplished. Special transaction mechanism to handling failures is designed that has the following properties.

S. Ambroszkiewicz (✉) · W. Bartyna · K. Skarzynski · M. Stepniak
Institute of Computer Science, Polish Academy of Sciences, Jana Kazimierza 5,
01-248 Warsaw, Poland
e-mail: sambrosz@ipipan.waw.pl

S. Ambroszkiewicz · W. Bartyna · K. Skarzynski · M. Stepniak
Systems Research Institute, Polish Academy of Sciences, Newelska 6,
01-447 Warsaw, Poland

P. Novais et al. (eds.), *Intelligent Distributed Computing IX*,
Studies in Computational Intelligence 616,
DOI 10.1007/978-3-319-25017-5_10

1. Failed services may be replaced with other services during task realization.
2. General plan may be changed.
3. The transaction ends after successful completion of the task, or inability to complete the task, or cancellation of the task.

The classic meaning of the term *transaction* in Information Technology goes back to the ACID properties of modifying a database. Transactions are understood as a mechanism for guaranteeing a group of operations (as a whole) performed on database to be atomic (either all or nothing), to produce consistent results, to be isolated from other operations, and their result to be durably recorded.

Long-running transactions avoid locks on non-local resources, use compensation to handle failures, potentially aggregate smaller ACID transactions (also referred to as atomic transactions), and typically use a coordinator to complete or abort the transaction. In contrast to rollback in ACID transactions, compensation restores the original state, or an equivalent, and is domain-specific. For example, the compensating action for a failure when transporting a cargo by one robot, is arranging a second robot that can continue the transport to the destination, and charging (as a penalty) the owner of the first robot for the delay.

OASIS Web Services Transaction (WS-TX) [2] is the standard specification that describes coordination types that are used with the extensible coordination framework described in the WS-Coordination specification. It defines two coordination types: Atomic Transaction and Business Activity. These coordination type can be used for building applications that require consistent agreement (transaction) on the outcome of distributed activities (services).

Based on the WS-TX standard the transaction protocols have been designed for multi-robot systems, and implemented in the Autero. The system has been tested in a universal simulated environment implemented in Unity 3D. Tests performed in a real environment are always limited by the devices (robots) and their limited range and capabilities. From the point of view of the proposed information technology (the protocols) the fact that the environment is simulated is irrelevant.

Since services are provided by heterogeneous robots, there must be common and generic representation (ontology) of the environment where the robots operate. The ontology consists of concepts and relations between them, i.e. objects, object attributes, and the relations between objects. Each object is of a certain, pre-defined type. The object type is defined by its attributes, and by the internal (hierarchical) structure, i.e. object may consist of sub-objects and relations between these sub-objects. An elementary type has no internal structure, so it is defined only by attributes. A complex type has a hierarchical structure of subtypes. The ontology is defined as a hierarchical collection of types of objects (see [3]). Primitive attributes and relations are the key elements for constructing the types. The object itself, as an instance of its type, is defined by assigning specific values to its attributes and by specifying relations. Primitive attributes and primitive relations must be *measurable* and *recognizable* by the robots.

The ontology is common for all components of the system, so that the robots are context-aware (see [4]). The ontology is also used to specify tasks, and define types of services called *service interfaces* consisting of the following elements.

- Type of service, i.e. type of action that the service performs.
- Specification of the inputs and outputs of the service.
- The conditions required for input of the service (preconditions), and the effects of its execution (postconditions) specifying the output. These conditions are expressed as relations between objects in the environment (ontology).
- Service attributes.

Service attributes contain information about the static features of a service and are used during planning, for example, operation range for a transport service, and average realization time.

The procedure of reserving the services for a task realization (according to a fixed plan) is called *arrangement* and has a form of contract between client and service provider. A similar solution was applied in the ASyMTRe-D approach [5].

There are several multi-robot systems architectures, see for example [6], [7], and [8]. However there, the tasks are executed in a tightly coupled manner and dedicated to a restricted class of tasks. There are also architectures that coordinate task execution, e.g. ALLIANCE [9] and M+ [10] with mechanisms for handling the failures. However, they are based on direct low level control that requires dedicated algorithms dependent on the robot hardware. Viewing a robot function as a service (having common interface) allows to apply Service Oriented Architecture (SOA) paradigm to multi-robot systems.

2 The Architecture of Autero

Autero was designed according to the SOA paradigm [11]. The system components communicate to each other using generic protocols. Repository stores ontology and provides access to it by the other system components. It also has a graphical user interface (GUI) for developing the ontology, and its management.

Task Manager (TM for short) represents a client, and provides a GUI for the client to define tasks and monitor their realization. The Planner provides abstract plans for TM, that are used to construct a concrete plan on the basis of information of available services (provided by Service Registry) and by arranging these services (by the Arrangement Module (AM) in a business process. TM controls the plan realization by communicating with the services arranged in the plan.

Arrangement is performed by sending to services requests (in the form of intentions) and collecting answers as quotes (commitments). Service Registry stores information about services currently available in the system. Each service, in order to be available, must be registered in Service Registry via Service Manager (SM). It is a robot interface for providing its services for an external client, in this case, TM is

the client. SM controls the execution of the subtask delegated by TM and reports success or failure to TM.

Task is defined as logical formula that describes the required final situation in the environment by using types, objects and relations from the ontology stored in Repository.

For a given task, Planner returns abstract plans, that when arranged and executed, may realize the final situation specified by the task in question. An abstract plan is represented as a directed graph where nodes are service types and edges correspond to causal relationship between the output of one service and the input of the second service. The relationships determine the order of arrangement and then the execution of a concrete plan that has also a form of a directed graph (called business process) however, its nodes are concrete arranged services. Sometimes it is not possible to arrange the whole business process before the start of execution phase. In this case, the business process includes the unassigned nodes for which the arrangement is to be done later on.

In a concrete plan its node may represent a composed service (as a subprocess) consisting of already arranged services. Plan may also include handlers responsible for a compensations and failure handling.

Task Manager initializes the service execution by sending the required input data to the Service Manager. The service is executed in accordance with the agreement made in the arrangement phase. After execution phase, Service Manager sends a response with the output data, being a confirmation of successful subtask completion, e.g. changing situation in the environment to the required one. Task Manager can also stop the service execution before its completion. This may be caused by the task cancellation by the client, a failure during concurrent execution of other services in the plan (that can not be replaced), or by changes in the environment making the plan infeasible.

Robot may not be able to successfully complete the task. In this case, its Service Manager notifies Task Manager by sending a detailed description of the problem. On this basis, TM can take appropriate actions. If Service Manager is not able to send such information, TM must invoke appropriate cognitive service (special patrolling robot, if available) to recognize the situation resulting from the failure.

3 Failure Handling

Failures during task accomplishing by robots may cause problems that must be handled. Mobile robotics is still a subject of extensive research, and at its current stage of development, failures occur frequently.

Task Manager is equipped with a failure handling mechanism based on the simple algorithm.

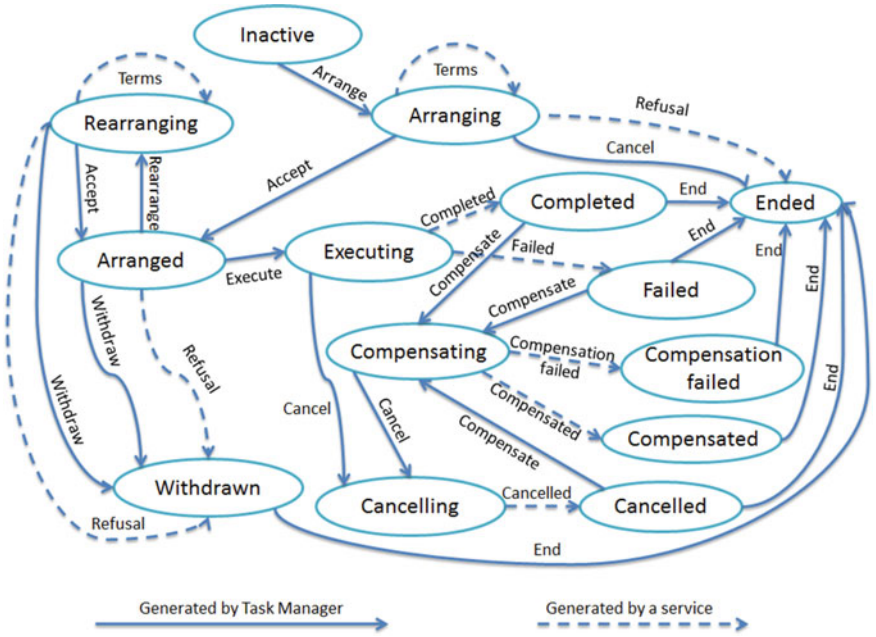

Fig. 1 Transaction protocol state transition diagram

All services are performed within a transaction that contains a dynamic set of participants. The transaction does not require all participants to successfully complete their tasks. The failure of a single service does not require the termination of the transaction. Termination is only necessary when it is not possible to continue the task.

Compensation is performed after a cancellation of a subtask execution by a service or the occurrence of a failure that interrupts the execution. It is designed to restore the original state of the environment before the execution. Since restoring that situation is sometimes impossible, the compensation may change the situation resulting from the failure to a situation from which the task realization can be continued. Note that even for such simple tasks (that seem to be trivial) a universal failure recovery mechanism and corresponding compensations are not easy to design and implement. A concrete plan should contain predefined procedures for failure handling and compensations.

Communication between Task Manager and Service Manager is done according to the transaction protocol which defines the states of the services and messages used to change them (see Fig. 1). It allows Task Manager to initialize particular phases of service invocation, monitor their progress, and perform additional actions, e.g. compensation. A service sends messages (according to the protocol) to notify Task Manager about the status of the delegated task performance.

All necessary data required for a task execution is a part of the transaction protocol message. This method allows to ensure the greater consistency of the system state. During the task execution messages are sent according to the specific sequences. They can create different combinations, but a set of possible messages in a given state of the service is strictly defined in the transaction protocol.

4 Conclusions

The presented work should be viewed as a preliminary study of the important and hard problem of designing mechanisms for handling failures and recoveries in open and heterogeneous multi-robot systems. The mechanisms must be based on generic protocols, so that problem is reduced to design such protocols.

The prototype system Autero verified that the proposed mechanisms of transactions are useful in the systems of heterogeneous robots in a simulated environment, and may improve their reliability. Task Manager can be configured to fully automate task accomplishment. Tasks can be delegated not only by human users but also by other system components or any software applications that can communicate according to the specified protocols.

Acknowledgments This work was carried out within the project RobREx—Autonomy for rescue and exploration robots, grant NRDC no PBS1/A3/8/2012. Marcin Stepniak was partially supported by the Foundation for Polish Science under International PhD Projects in Intelligent Computing.

References

1. Okada, K., Ogura, T., Haneda, A., Fujimoto, J., Gravot, F., Inaba, M.: Humanoid motion generation system on hrp2-jsk for daily life environment. In: 2005 IEEE International Conference on Mechatronics and Automation, vol. 4, pp. 1772–1777. doi:10.1109/ICMA.2005.1626828 (2005)
2. WS-TX 1.2 OASIS Standards.: https://www.oasis-open.org/committees/ws-tx/ (2009). Accessed 02 February 2009
3. Ambroszkiewicz, S., Bartyna, W., Faderewski, M., Terlikowski, G.: Multirobot system architecture: environment representation and protocols. Bull. Polish Acad. Sci. Tech. Sci. **58**(1), 3–13 (2010). doi:10.2478/v10175-010-0001-y
4. Go, Y.C., Sohn, J.C.: Context modeling for intelligent robot services using rule and ontology. In: The 7th International Conference on Advanced Communication Technology, ICACT 2005. vol. 2, pp. 813–816. IEEE (2005)
5. Tang, F., Parker, L.E.: A complete methodology for generating multi-robot task solutions using asymtre-d and market-based task allocation. In: 2007 IEEE International Conference on Robotics and Automation, pp. 3351–3358. IEEE (2007)
6. Long, M., Gage, A., Murphy, R., Valavanis, K.: Application of the distributed field robot architecture to a simulated demining task. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005. pp. 3193–3200. IEEE (2005)

7. Michael, N., Zavlanos, M.M., Kumar, V., Pappas, G.J.: Distributed multi-robot task assignment and formation control. In: IEEE International Conference on Robotics and Automation, ICRA 2008. pp. 128–133. IEEE (2008)
8. Pagello, E., Ferrari, C., d'Angelo, A., Montesello, F.: Intelligent multirobot systems performing cooperative tasks. In: 1999 IEEE International Conference on Systems, Man, and Cybernetics, IEEE SMC'99 Conference Proceedings. vol. 4, pp. 754–760. IEEE (1999)
9. Parker, L.E.: Alliance: an architecture for fault tolerant multirobot cooperation. IEEE Trans Robot Autom **14**(2), 220–240 (1998)
10. Botelho, S.C., Alami, R.: M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement. In: Proceedings of IEEE International Conference on Robotics and Automation, 1999. vol. 2, pp. 1234–1239. IEEE (1999)
11. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: service-oriented architecture best practices. Prentice Hall Professional, Upper Saddle River (2005)