# KAoS: A Policy and Domain Services Framework for Grid Computing and Semantic Web Services

Andrzej Uszok, Jeffrey M. Bradshaw, and Renia Jeffers

Institute for Human and Machine Cognition (IHMC)
40 S. Alcaniz, Pensacola, FL 32502
{auszok, jbradshaw, rjeffers}@ihmc.us
http://www.ihmc.us

In this article we introduce KAoS, a policy and domain services framework based on W3C's OWL ontology language. KAoS was developed in response to the challenges presented by emerging semantic application requirements for infrastructure, especially in the area of security and trust management. The KAoS architecture, ontologies, policy representation, management and disclosure mechanisms are described. KAoS enables the specification and enforcement of both authorization and obligation policies. The use of ontologies as a source of policy vocabulary enables its extensibility. KAoS has been adapted for use in several applications and deployment platforms. We briefly describe its integration with the Globus Grid Computing environment.

## 1 Introduction

Policy is an essential component of automatic trust systems [7]. Advances in Web Services (*http://www.w3.org/2002/ws*), Grid Computing (*http://www.gridforum.org*), P2P networks (*http://www.afternapster.com/*), Semantic Web Services (*http://www.swsi.org/*) and the convergence of all these environments creates a need for a highly adaptable, semantically rich policy mechanisms supporting the establishment of trust in all its aspects. OWL (*http://www.w3.org/2001/sw/WebOnt*), based on Description Logic [1], is an emerging standard for semantically rich services infrastructure that can be used effectively not only by people but also by software agents that represent them. Trust systems of the future will need to be able to recognize and reason about semantics used by services and agents; thus OWL is a natural choice for the development of next-generation policy service components of trust systems that will be up to the challenge of the Semantic Web. The KAoS policy and domain services framework [2, 3, 8] uses OWL both to represent policies, domains and other managed entities, and to describe their elements. The use of OWL enables flexible extension of the framework architecture, consistent with the advanced requirements of semantic applications.

## 2 KAoS Services Framework Architecture

KAoS is a collection of componentized services compatible with several popular agent platforms, including the DARPA CoABS Grid [9], the DARPA ALP/Ultra*Log Cougaar agent framework (*http://www.cougaar.net*), CORBA (*http://www.omg.org*) and Brahms [6]. The adaptability of KAoS is due in large part to its pluggable infrastructure based on Sun's Java Agent Services (JAS) (*http://www.java-agent.org*). While initially oriented to the dynamic and complex requirements of software agent applications, KAoS services have also been adapted to general-purpose grid computing [10] and Web Services [9] environments.

Under DARPA and NASA sponsorship, we have been developing the KAoS policy and domain services to increase the assurance and trust with which agents can be deployed in a wide variety of operational settings. *KAoS Domain Services* provide the capability for groups of software components, people, resources, and other entities to be semantically described and structured into organizations of domains and subdomains to facilitate collaboration and external policy administration. *KAoS Policy Services* allow for the specification, management, conflict resolution, and enforcement of policies within domains.

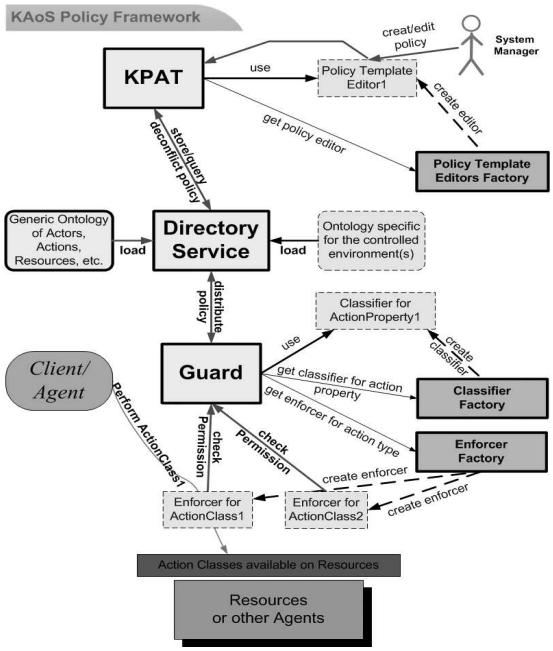


**Fig. 1.** Selected elements of the KAoS policy and domain services framework.

Figure 1 presents basic elements of the KAoS framework. Framework[1] functionality can be divided into two categories: generic and application/platform-specific. The generic functionality includes reusable capabilities for:

- Creating and managing the set of core ontologies;
- Storing, deconflicting and querying;
- Distributing and enforcing policies;
- Disclosing policies.

For specific applications and platforms, the KAoS framework can be extended and specialized by:

- Defining new ontologies describing application-specific and platform-specific entities and relevant action types;
- Creating extension plug-ins specific for a given application environment such as:
  - Policy Template and Custom Action Property editors;
  - Enforcers controlling, monitoring, or facilitating subclasses of actions;
  - Classifiers to determine if a given instance of an entity is in the scope of a given class-defining range.

## 3 KAoS Ontologies

The current version of the core KAoS Ontologies (*http://ontology.ihmc.us/)* defines basic concepts for actions, actors, groups, places, various entities related to actions (e.g., computing resources), and policies. It includes more than 100 classes and 60 properties.

The core *actor ontology* contains classes of people and software components that can be the subject of policy. Groups of actors or other entities may be distinguished according to whether the set of members is defined extensionally (i.e., through explicit enumeration in some kind of registry) or intentionally (i.e., by virtue of some common property such as types of credentials actors possess, or a given place where various entities may be currently located).

The core *action ontology* defines various types of basic actions such as accessing, communication, monitoring, moving, and so forth. An ontological definition of an action associates with it a list of properties describing context of this action or a current state of the system relevant to this action. Example properties of action classes are, for instance: destination of the communication, type of encryption used, resources accessed, time, previous history, and so forth. Each property is associated with the definition of a range of values it could have for each of the action classes. A particular instance of the action class can take values on the given property only from within this range. Actions are also divided into *ordinary actions* and *policy actions,* the latter

[1] Figure 1 emphasizes infrastructure supporting the specification and use of authorization policies. There are additional components to support obligation policies and other aspects of the system but they were omitted from the picture for simplicity and paper size restriction.

comprising those actions that have to do with the operations of the KAoS services themselves[2].

For a given application, the core KAoS ontologies are usually further extended with additional classes, individuals, and rules, which use the concepts defined in the core ontologies as superconcepts. This allows the framework to discover specialized concepts by querying an ontology repository for subclasses or subproperties of the given concept or property from the core ontologies. For example additional application-related context could be added to actions such as specific credentials used in a given environment.

During the initialization process, the core policy ontologies are loaded into the *KAoS Directory Service* using the namespace management capabilities of the *KAoS Policy Administration Tool (KPAT)* graphical user interface. Additional application-specific or platform-specific ontologies can then be loaded dynamically using KPAT or programmatically using the appropriate Java method. A distributed version of the KAoS Directory Service is currently being implemented. We are also studying possibilities for interaction among multiple instances of Policy Services [9].

The Directory Service is also informed about the structure of policies, domains, actors, and other application entities. This information is added to the ontology repository as instances of concepts defined in pre-loaded ontologies or values of these instance properties. As the end-user application executes, instances relating to application entities are added and deleted as appropriate.

KAoS employs the Jena Semantic Web Toolkit by HP Labs in Bristol (*http://www.hpl.hp.com/semweb*) to incrementally build OWL definitions and to assert them into the ontology repository managed by the Directory Service. In order to provide description logic reasoning on the OWL defined ontologies, the Java Theorem Prover (*http://www.ksl.stanford.edu/software/JTP*) inference engine has been integrated with KAoS. Performance is always an issue in logic reasoning; however, the steady improvement of JTP has led to a dramatic increase in its performance—an order of magnitude or more in some cases—in the last two years. The most time consuming operation in JTP is asserting new information, which happens mostly during system bootstrap. Currently, loading of the KAoS core ontologies takes less than 16 seconds on Pentium III 1.20 GHz with 640 MB RAM. Adding a policy takes usually less than 340ms. Querying JTP about ontology concepts and policies is much faster and takes only a few milliseconds.

## 4 Policy Representation

In KAoS, policies can express authorization (i.e., constraints that permit or forbid some action) or obligation (i.e., constraints that require some action to be performed, or else serve to waive such a requirement) for some type of action performed by one or more actors in some situation [2]. Whether or not a policy is currently applicable may be conditional upon some aspect of the situation. Auxiliary information may be associated with a policy, such as a rationale for its existence or a specification of

[2] This distinction allows reasoning about actions on policies and the policy framework without resorting to the use of special "metapolicy" mechanisms.

some penalty for policy violation. In contrast to many existing policy systems [4; *http://www.policy-workshop.org*], KAoS aims at supporting both an extensible vocabulary describing concepts of the controlled environment and also an evolution of its policy syntax. Such features are one beneficial consequence of defining policies within ontologies and using an extensible framework architecture [11].

In KAoS, a policy is represented as an ontology instance[3] of one of the four types of policy classes: positive or negative authorization, and positive or negative obligation. The instance possesses values for various management-related properties (e.g., priority, time stamp, site of enforcement) that determine how the given policy is handled within the system. The most important property value is the name of a controlled action class, which is used to determine the actual meaning of the policy. Authorization policies use it to specify the action being authorized or forbidden. Obligation policies use it to specify the action being obliged or waived. Additionally the controlled action class contains a trigger value that creates the obligation, which is also a name of the appropriate class of actions. Policy penalty properties contain a value that corresponds to a class of actions to be taken following a policy violation.

As seen from this description, the concept of action is central to the definition of KAoS Policy. Typically any action classes required to support a new policy are generated automatically by KAoS when a user defines new policy (usually using KPAT). Through various property restrictions, a given subject of the action can be variously scoped, for example, either to individual agents, to agents of a given class or to agents belonging to a particular group, and so forth. The specific contexts in which the policy constraint applies can be precisely described by restricting values of the action's properties, for instance requiring that a given action be signed using an algorithm from the specified group.

## 5 Policy Management

The real strength of KAoS is in its extensive support for policy life-cycle management. KAoS hides many elements of complexity of this process from the user. KAoS also provides a sophisticated policy disclosure interface enabling querying about policy impact on planned or executed actions.

### 5.1 Graphical Interface to Ontology Concepts

The KPAT graphical interface to policy management hides the complexity of the OWL representation from users. The reasoning and representation capabilities of OWL are used to full advantage to make the process as simple as possible. Whenever a user has to provide an input is always presented with a complete set of values he can choose from, which are valid in the given context.

As in the case of the generic policy editor shown on figure 2, a user, after selecting an actor for a new policy, is first presented with the list of actions the given type of

[3] See *http://ontology.ihmc.us/SemanticServices/S-F/Example/* for an example of KAoS policy syntax.

actors is capable to perform based on the definition in the ontology relating actions to actors by the *performedBy* property. When the user selects a particular action type information about all the properties, which can be associated with the given actions, are presented. For each of the properties, the range of possible values is obtained; instances and classes falling into this range are gathered if the user wants to build a restriction on the given property, thus narrowing the action class used in the build policy to its context.

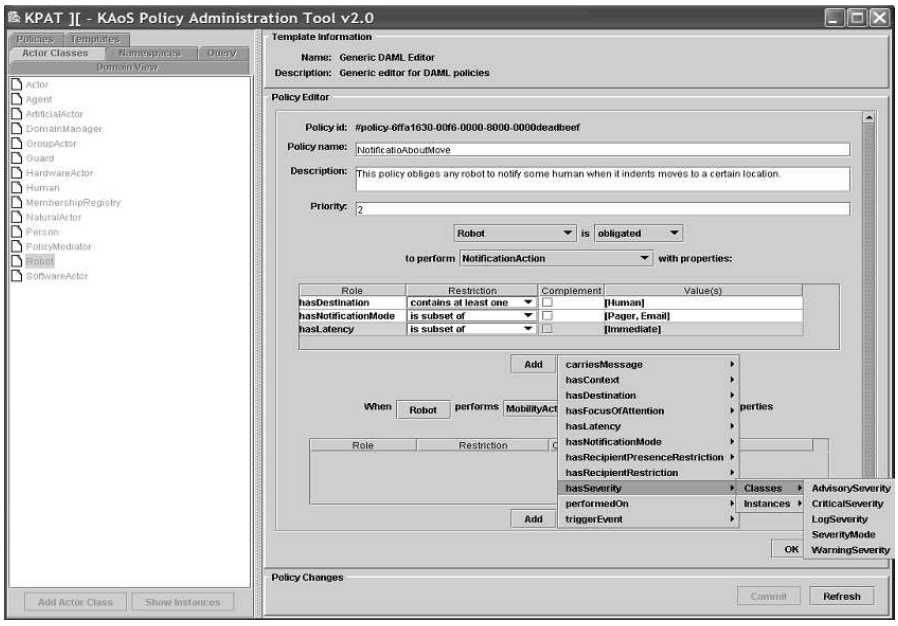


**Fig. 2.** KPAT generic policy builder – an example of ontology-guided interface.

### 5.2 Policy Administration

Each time a new policy is added or an existing one is deleted or modified, the potential impact goes beyond the single policy change. Policy administrators need to be able to understand such interactions and make sure that any unwanted side effects are eliminated. KAoS assists administrators by identifying instances of given types of policy interactions, visualizing them, and, if desired, facilitating any necessary modifications.

One important type of interaction is a policy conflict [2, 8]. For example, one policy might authorize actor A to communicate with any actor in group B while a new policy might forbid actor A from communicating with actor B1, a member of B. In general, if a new policy overlaps in key properties of a subset of controlled actions with an existing policy of a potentially conflicting modality (i.e., positive vs. negative authorization (as in our example); positive vs. negative obligation; positive obligation

vs. negative authorization), some means must be used to identify the conflict and to determine, in the area of overlap, which policy takes precedence[4]. If precedence cannot be determined otherwise, KAoS will ask the administrator to determine the appropriate action (Fig. 3).

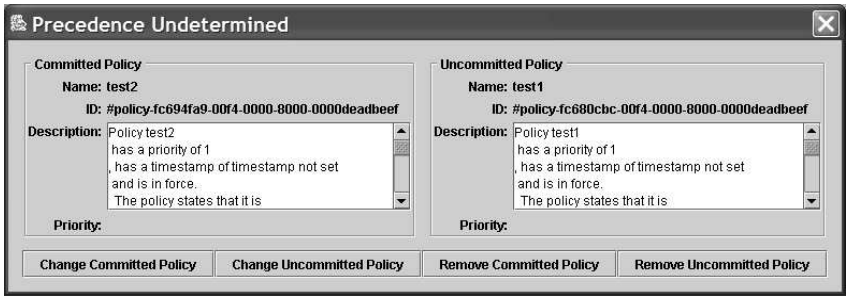


**Fig. 3.** Notification about policy conflict and options available to the administrator.

The following policy actions can be performed on a pair of overlapping policies:

- *Remove Policy*: one of the overlapping policies can be completely removed;
- *Change Priority*: priorities of the policies can be modify so they either do not conflict or they alter the precedence relation[5];
- *Harmonize Policy*: the controlled action of the selected overlapping policy can be modified using an automatic harmonization algorithm to eliminate their overlap; see [8] for details. This required modification of the restrictions in of the policy controlled actions by building either intersection (by using *owl:intersectionOf*) or differences (by using *owl:complementOf*) of the previous ranges in the two conflicting policies.
- *Split Policy*: the controlled action of the selected overlapping policy can be automatically split into two parts: one part that overlaps with the other policy and the other which does not. Then the priorities of these parts can be modified independently. The splitting algorithm is similar to the harmonization and is currently in development.

In the future, a more sophisticated user interface will allow for modification of entire sets of policies at once.

Whereas the goal of policy conflict resolution is to ensure consistency among the policies in force, other forms of analysis are needed to ensure policy enforceability. In

[4] If desired, precedence relations can be predefined in the ontology, permitting partially or totally automated conflict resolution.

[5] We currently rely exclusively on the combination of numeric policy priorities and update times to determine precedence—the larger the integer and the more recent the update the greater the priority. In the future we intend to allow people additional flexibility in designing the nature and scope of precedence conditions. For example, it would be possible to define default precedence over some policy scope based on the relative authorities of the individual who defined or imposed the policies in conflict, which policy was defined first, and so forth.

some cases, the implementation of policy may be impossible due to prior obligations of the actor or oversubscription of resources. In the future, KAoS will be able to suggest ways of relaxing such non satisfy constraints in certain situations.

In some cases, two complementary policies of the same modality can create unanticipated problems. For example, one policy may prevent communication among actors within domain A while another policy might prevent communication to actors outside of the domain. Though the two policies would not conflict, their combination would result in the inability of actors in domain A to communicate at all. It should be possible in the future to flag these and other situations of potential interest to administrators.

### 5.3 Policy Exploration and Disclosure

A human user or software component uses KAoS to investigate how policies affect actions in the environment. In general, the answers to these queries are decided by inferring whether some concrete action falls into a category of action controlled by one or more policies, and then determining what conclusions about the described action can be drawn. As part of KAoS policy exploration and disclosure interfaces we provide the following kinds of functionality:

- *Test Permission:* determine whether the described action is permitted.
- *Get Obligations:* determine which actions, if any that would be obligated as a follow on to some potential action or event. For instance, there might be an obligation policy which specified that if an actor were to receive information about a particular topic then the system would be obligated to log or forward this information to some other party.
- *Learn Options:* determine which policy-relevant actions are available or not available in a given context. For example, the actor may specify a partial action description and KAoS would return any missing (required) elements of the action with ranges of possible values—for instance, information about missing credentials.
- *Make Compliant:* transform the action an actor tries to perform from a policy non-compliant to a policy-compliant one by informing it about the required changes that would need to be made to the action based on existing policies. For instance, if the system attempted to send a message about particular subject to a few actors, the list of actors might need to be trimmed to some subset of those actors or else extended to include some required recipients. Or else maybe the content of a message would need to be transformed by stripping off sensitive information, and so forth.
- *Get Consequences:* determines the consequences of some action by observing and investigating possible actions in the situation created by a completion of the considered action(s) to the specified depth (consequences of consequences). This option has many variants currently under investigation.

### 5.4 Adapting Policy to Legacy Systems

When policy leaves the Directory Service, for performance reasons it typically has to map OWL into a format that is compatible with the legacy system with which it is being integrated. KAoS communicates information from OWL to the outside world by mapping ontology properties to the name of the class defining its range as well to a list with cached instances of that class that were in existence when the policy left the Directory Service. A particular system can use the cached instance for its computation; also in any moment it can refresh the list by contacting the Directory Service and providing the name of the range. Alternatively, the Directory Service can push changes to the system as they occur.

## 6. KAoS Policy Applications

KAoS is used in several applications ranging from the human-robotic teamwork for NASA and the Office of Naval Research [12], through massive societies of agents in the DARPA Ultralog project building next generation army logistic system, to Semantic Web Services interaction in the DARPA CoSAR-TS project [9][6]. Here, we briefly present a summary of how KAoS has been integrated with Grid Computing services on the Globus platform (*http://www.globus.org/*) [10].

Globus provides effective resource management, authentication and local resource control for the grid-computing environment, but has a need for domain and policy services. KAoS seemed to be a perfect complement to the Globus system, providing a wide range of policy management capabilities that rely on platform-specific enforcement mechanisms. By providing an interface between the Globus Grid and KAoS, we enable the use of KAoS mechanisms to manage GSI (Grid Security Infrastructure) enabled Grid services. GSI was the only component of the GT3 (Globus Toolkit) we used in the integration. The interface itself is a Grid service, which we called a KAoS Grid service. It provides Grid clients and services the ability to register with KAoS services, and to check weather a given action is authorized or not based on current policies. The clients or resources use their credential to request to be registered into one or more KAoS managed domains. The credential is a standard X.509 certificate that Globus uses for authentication. The credential is verified using the GT GSI. If the certificate is valid the registration request is sent to KAoS for registration into the desired domains. If the resource uses an application specific ontology to describe its capabilities, it will have to be loaded into the KAoS ontology using a utility provided by KAoS. Inside the KAoS Grid service, the registration is handled through the associated Guard. This allows KAoS to distribute all applicable policies to the appropriate Guard and enforce them. We plan to continue to enhance this service and port it to GT4 when it is available.

[6] See *http://ontology.ihmc.us/applications.html* for more information about these applications.

## 7 Conclusions

Originally KAoS was not tailored for trust negotiation and management. However, from the very beginning the architecture of the framework architecture and its extensive use of ontologies ensured its versatility and adaptability. It already provides most of the generic mechanisms enumerated as required for the policy system to be integrated with a trust system, as enumerated in [7].[7] Current work in the area of Semantic Web Services [9] will fill the gaps in the area of negotiation, necessary ontologies for credentials, and integration with existing PKI infrastructure. Our current prototypes integrating KAoS with grid computing security and credential mechanisms, as presented above, gives us confidence in the promise of future work in this same direction.

**Acknowledgments**
The authors gratefully acknowledge the sponsorship of this research by the NASA Cross-Enterprise and Intelligent Systems Programs, and a joint NASA-DARPA ITAC grant. Additional support was provided by DARPA's CoABS, Ultra*Log, and CoSAR-TS programs. Thanks to the other members of the KAoS project team: Maggie Breedy, Larry Bunch, Matthew Johnson, Hyuckchul Jung, Shri Kulkarni, James Lott, William Taysom, and Gianluca Tonti. We are also grateful for the contributions of Austin Tate, Pat Hayes, Niranjan Suri, Paul Feltovich, Richard Fikes, Jessica Jenkins, Rich Feiertag, Timothy Redmond, Sue Rho, Ken Ford, Mark Greaves, Jack Hansen, James Allen, and Robert Hoffman.

## References

[1] Baader, F., Calvanese, D., McGuinness, D., Nardi, D. & Patel-Schneider, P. (Ed.) (2003). The Description Logic Handbook. Cambridge University Press,
[2] Bradshaw, J. M., Beautement, P., Breedy, M. R., Bunch, L., Drakunov, S. V., Feltovich, P., Hoffman, R. R., Jeffers, R., Johnson, M., Kulkarni, S., Lott, J., Raj, A. K., Suri, N., & Uszok, A. (2003). Making agents acceptable to people. In N. Zhong and J. Liu (Eds.), Handbook of Intelligent Information Technology. Amsterdam: IOS Press, in press.
[3] Bradshaw, J. M., Dutfield, S., Benoit, P., & Woolley, J. D. (1997). KAoS: Toward an industrial-strength generic agent architecture. In J. M. Bradshaw (Ed.), Software Agents. (pp. 375-418). Cambridge, MA: AAAI Press/The MIT Press.
[4] Damianou, N., Dulay, N., Lupu, E. C., & Sloman, M. S. (2000). Ponder: A Language for Specifying Security and Management Policies for Distributed Systems, Version 2.3., Imperial College, London,
[5] Kahn, M., & Cicalese, C. (2001). CoABS Grid Scalability Experiments. O. F. Rana (Ed.), Second International Workshop on Infrastructure for Scalable Multi-Agent Systems at the Fifth International Conference on Autonomous Agents. ACM Press,
[6] Sierhuis, M. (2002). Brahms - Modeling and Simulating Work Practice. Univ. of Amsterdam Press,

[7] We hope to be able to integrate with Seamon's work on TrustBuilder in the future.

[7] Seamons, K., Winslett, M., Yu, T., Smith, B., Child, E., Jacobson, J., Mills, H. and Yu L. (2002). Requirements for Policy Languages for Trust Negotiation. In Proceedings of IEEE Workshop on Policy 2002.

[8] Uszok, A., Bradshaw, J., Jeffers, R., Suri, N., Hayes, P., Breedy, M., Bunch, L., Johnson, M., Kulkarni, S. and Lott, J. (2003). KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction and Enforcement. In Proceedings of IEEE Workshop on Policy 2003.

[9] Uszok, A., Bradshaw, J., Jeffers, R., Johnson, M., Tate A., Dalton, J., Aitken, S. (2004). Policy and Contract Management for Semantic Web Services. In Proceedings of the AAAI Spring Symposium on Semantic Web Services.

[10] Johnson, M., Chang, P., Jeffers, R., Bradshaw, J.M., Soo, V-W., Breedy, M. R., Bunch, L., Kulkarni, S., Lott, J., Suri, N., & Uszok, A. (2003). KAoS semantic policy and domain services: An application of DAML to Web services-based grid architectures. In Proceedings of the AAMAS 03 Workshop on Web Services and Agent-Based Engineering. Melbourne, Australia, July. (To appear in a forthcoming volume from Kluwer.)

[11] Tonti, G., Bradshaw, J. M., Jeffers, R., Montanari, R., Suri, N., & Uszok, A. (2003). Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. In D. Fensel, K. Sycara, & J. Mylopoulos (Ed.), The Semantic Web-ISWC 2003. Proceedings of the Second International Semantic Web Conference, Sanibel Island, Florida, USA, October 2003, LNCS 2870. (pp. 419-437). Berlin: Springer.

[12] Sierhuis, M., Bradshaw, J. M., Acquisti, A., Van Hoof, R., Jeffers, R., & Uszok, A. (2003). Human-agent teamwork and adjustable autonomy in practice. Proceedings of the Seventh International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS). Nara, Japan, 19-23 May.