A Functional and Behavioral Knowledge-based Implementation for Intelligent Sensors/Actuators

R.Dapoigny and **P.Barlatier** and **L.Foulloy** and **E.Benoit**
LISTIC-ESIA
B.P.806 ANNECY cedex (France)
Phone: +33 450 096529 Fax: +33 450 096559
{richard.dapoigny,patrick.barlatier,laurent.foulloy,eric.benoit}@univ-savoie.fr

Abstract

As the complexity of industrial processes increases, it requires the use of intelligent sensors or actuators (known as Intelligent Instruments) to allow for more comprehensive and efficient behavior of the system to be monitored. These components achieve some global goal called a service through the use of a set of elementary actions (internal services). A challenging task in networked intelligent sensors/actuators, is to allow for automated interactions between arbitrary services on networked components. To resolve this task, two successive problems emerge, first the design of consistent control systems with appropriate knowledge and second the monitoring which concerns the dynamic behavior of such distributed control systems. In this paper, we focus on the first aspect and provide an implementation of a knowledge representation model where services can be structured and related via a mereo-topological approach. The domain ontologies include a system ontology which is application-dependent, a functional ontology which relies on teleology and a behavioral mereotopology extending the functional mereology. To support the behavior representation, we propose a GUI based on a modified existing formalism (DAML-S). We discuss the knowledge representations in depth and present the design supporting system centered on the mereo-topological approach for intelligent sensors and actuators

Introduction

Technological advances in the areas of industrial networks and micro-controller capabilities promote the design of systems composed of smart components called Intelligent Instruments (II). Their fundamental property is the inclusion of an active processing element (a micro-controller) which encapsulates a number of functionalities, known as external services, and that are accessible through a network interface via perceptual channels Bouras & Staroswiecki (1997). In the modelling of artificial systems or natural systems, the knowledge about the system can be divided from an epistemologic point of view, into three parts, knowledge about causality, knowledge about relationships and knowledge about functionality Bruegge & Dutoit (1999). The knowledge about relationships also called structural model describes the entities and their relationships, while the knowledge about functionality focusses on formal specifications such as goals, pre and post-conditions. In II, each instrument manages information describing the knowledge concerning the physical system which is interacting with the instruments and the knowledge about the instruments themselves including their functional capabilities defined in the design process and the effective behavior emerging from the runtime process Dapoigny, Benoit, & Foulloy (2003). Therefore, in this paper we tackle the following design oriented problem: given a physical system related to an II, we check the user-defined functions that can achieve a desired behavior using the knowledge of its structure and initial dedicated functions. An integrated knowledge-based system centered on mereo-topology is implemented through a graphical tool (GCAPTOOL) which requires users specifications of an instrument at a functional level and translates them into a set of services, internal services, events and functioning modes. A coherent and complete knowledge database has been investigated. It is centered both on a small number of concepts that will serve as a basis for a behavioral analysis and on a structured knowledge with a mereo-topological approach intended to support intelligent behavior. Section 2, introduces the basic features of intelligent instruments. Section 3 discusses the role of major concepts in the area of intelligent sensor/actuators. Section 4 proposes rough outlines of a knowledge-based mereo-topology including functional and behavioral requirements. Implementation is reported in section 5.

Intelligent Instruments

Formally, an II is seen as an artifact related to a chunk of engineering process allowing two processes to coexist concurrently, i.e., a physical process and a control process. We presuppose that the physical process evolves more slowly that the control process (real-time condition). The control process takes into account both meta actions which are related to the physical process states (corresponding to measurements) and actions on the process itself. These two conceptual levels have been pointed out by several authors such as Spalazzi & Traverso (1999)Toppano (2002) and must be included into the knowledge modelling. In each service, a

sequence of internal services which are linked through an ordered graph acts like a plan. For the design of intelligent instruments, dedicated instrument functionalities (describing the interface control unit-physical sensor/actuator) are implemented in basic internal services. The instruments designer only focusses on external service composed with a set of internal services. A basic external model of an intelligent instrument was proposed, based on services and operating modes Benoit, Foulloy, & Tailland (2001). Services are organized into subsets called "USer Operating Modes" (USOM). In this model, a sensor service can be required, only if the current active USOM includes this service. This prevents requiring services when they can not be available. Unfortunately, this model is unable to provide automated interactions between arbitrary allocated services on distributed instruments, and the static graph appears to be inappropriate to reason about dynamic knowledge. Therefore, a knowledge-based model has been proposed to extend the previous one.

Basis of the Knowledge-based Model

This section gives outlines of the knowledge-based model dedicated to intelligent sensors/actuators. The analysis is limited to the class of artefacts whose behavior can be interpreted in terms of flow structures such as energy or information. In related areas such as Process Knowledge Bases Dooley, Skilton, & Anderson (1998), the authors suggest that complete information about an engineering or computer information system is expressed with three types of knowledge (i.e., structural, functional and behavioral knowledge). We adopt this representation in distributed instrumentation where the knowledge about the system will be divided into three concepts and their related knowledge:

- structural knowledge – knowledge about physical objects and their physical relations
- functional knowledge – knowledge about the purpose and the way the physical role of objects are used.
- behavioral knowledge – knowledge about the changes in the environment and relations between states and events.

From the three basic concepts, a set of five sub-concepts can be identified, i.e. properties, physical components, physical role (or physical quantity), goal and action. The design step is primarily concerned with the structural and functional knowledge, whereas the dynamic aspect is more related to the behavioral knowledge. The structural knowledge is represented with a system mereo-topology which describes the physical environment of sensors and (or) actuators. This mereo-topology highlights both the concepts of physical entity through a mereology and the concept of physical role (or quantity) which connects two physical entities iff there exists an energy path between them Borst *et al.* (1995). This rule allows for dependence relations between variables through their semantic representation. For the functional and behavioral knowledge, we investigate a functional ontology which is extended with topological features (pre-conditions including event occurrences) to build a behavioral mereo-topology. This ontology provides the

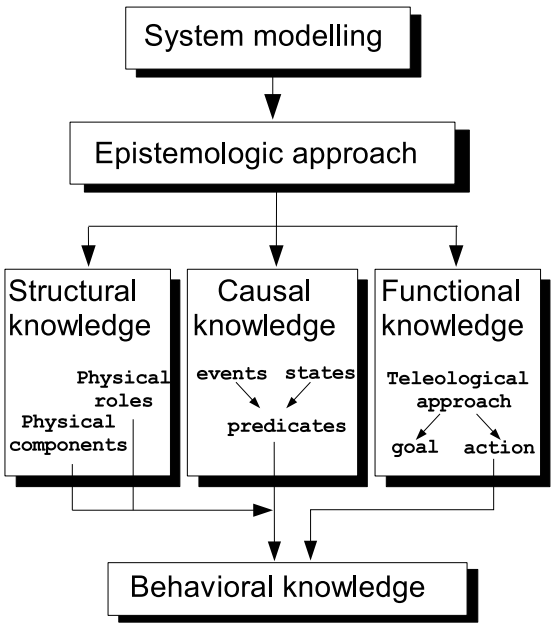

Figure 1: Concepts for Intelligent Instruments

ability to reason automatically with functional models based on their goals and on their way to achieve them. The relation between structural and functional ontologies relies on the variable semantics. While the mereology is the qualitative formalization of the part-of relationship between entities, the topology deals with the connection relationship Simons (1987). As suggested in Salustri (1998), mereology and topology can be applied to functional knowledge and, as a consequence, they seem to have obvious application to the modelling of instruments. First, mereology and topology promote structured analysis and provide a framework that is as significant as problem complexity increases. Second, they help prune the search space of possible functions by eliminating functions that are logically inconsistent with attainable goals. Third, as a network of logical relations results from mereology and topology, more sophisticated decision process can be explored at run-time based on dependence relations. Fourth, these theories can be embedded into improved computer-based tools. The major advantage is that end-users will have increasing time to focus on aspects of new functionalities development and design for which the human mind is more suited.

The Mereo-topological Representation

Representation of states

We consider variable representation as physical entity and physical role which is related to the physical quantity defined in standard ontologies Gruber & Olsen (1994) or to the modelling law describing the physical mechanism. Let us consider the sort of individuals denoting the physical entities of the physical system related to the instrument, the sort of physical roles which can be either physical quantities

or control/monitoring roles, the sort of values (either real or linguistic constants), the sort of predicates and their respective associated sets, $\oplus$, $\mathcal{R}$, $\mathcal{V}$ and $\mathcal{P}$.

Definition 1 *In a functioning mode referred by i, a local state s_i represents a set of properties at a given time t, where each property is expressed as a predicate such as :*

$$s_i = \bigwedge_j \Pi_{ij} \quad \texttt{where} \quad \Pi_{ij} \in \mathcal{P}^n \tag{1}$$

each predicate Π_{ij} denotes an atomic state and n refers to the maximum number of predicates in the mode i.

With such a knowledge representation, the state is unique, that is to say, when switching between two different states, at least one property must change. Examples of atomic and local states :

$\mathtt{s_{m1}} = \mathtt{P_1} = \mathtt{Holds}(\mathtt{x\_spectral\_sensitivity}, \mathtt{sample}, 0.35)$

$\mathtt{s_{m2}} = \mathtt{P_3} \wedge \mathtt{P_4} \wedge \neg \mathtt{P_2} = \mathtt{Open}(\mathtt{position}, \mathtt{Valve}) \wedge \mathtt{Greater\ Than}(\mathtt{speed}, \mathtt{speed0}, \mathtt{water1}) \wedge \neg \mathtt{Achieved}(\mathtt{to\_compute\_speed}, \mathtt{I1})$

Functional knowledge

Functions must contain device-independent knowledge that will be used to determine potential behavior (normal, degraded or out_of_order). This assumption facilitates the construction of knowledge databases and allows to classify functions relatively to the designers intents. In the context of intelligent instruments, assuming the teleological interpretation, functions must represent the functional role that supports a given physical process. extracted from functional patterns dedicated to specific instruments or defined by the end user. Any functional concept will be described by a (sub-)goal definition which is related to the intensional aspect of function Lind (1994) and some possible actions (at least one) in order to fulfill the intended (sub-)goal Hertzberg & Thiebaux (1994)Lifschitz (1993). The functional modelling requires first to define how functional concepts (i.e., goals and actions) are related, and secondly to describe their representation (i.e., data structures).

- Unlike general framework where goals cannot be formalized and relationships among them cannot be semantically captured, the present framework restricted to engineering physical entities makes it possible to describe a mereology of goals. As a consequence, an individual f is a functional part of a whole F iff its subgoal is required to achieve the upper goal (property ϕ). Complex goals can be expressed as a mereological fusion of atomic goals. In the context of mereology, a satisfied predicate ϕ over entities f defines the mereological fusion (or mereological sum) of those entities, if it exists, as the thing which overlaps all and only those things which overlap some f such that ϕ Varzi (1996)Smith (1996). Some important consequences are that empty sums do not exist (a given instrument has at least one definite goal). One of the major benefits of mereological framework is that it allows for different abstraction level to appear in the same model. The effective actions which implement the goal intents are performed by C or Java computing functions.
- Representation of functions as *"to do X"* has been used by several researchers Umeda & al. (1996)Miles (1981) and we extend that textual definition of the function goal in the following way:

  Definition 2 *An atomic goal represents the mapping $\mathcal{V}_A \times \mathcal{R}^n \times \Phi^n \rightarrow \mathcal{G}$ as the tuple $g =< v_A, \{r_i, \phi_i\} >$, $v_A \in \mathcal{V}_A, r_i \in \mathcal{R}, \phi_i \in \Phi$ and $g \in \mathcal{G}$ where $\mathcal{V}_A$, $\mathcal{R}$, Φ and $\mathcal{G}$ are the sets associated with the respective sorts of verbs describing elementary actions, the sort of physical roles (physical quantities or control/monitoring roles), the sort of physical entities of the physical system related to the instrument, and the sort of elementary sub-goals.*

  The integer value n denotes the amount of pairs $\{r_i, \phi_i\}$ involved in a given goal.

In such a way, a functional model of the physical system is built by associating the teleological part of functions with the structural part, i.e. the physical components of the system. Examples of atomic sub-goals are:

$\mathtt{g1} =< \mathtt{To\_acquire}, \mathtt{x\_spectral\_sensitivity}, \mathtt{sample} >$

$\mathtt{g2} =< \mathtt{To\_act\_upon}, \mathtt{position}, \mathtt{Valve} >$

$\mathtt{g3} =< \mathtt{To\_act\_upon}, \mathtt{angle}, \mathtt{axis} >$

With a limited set of appropriate elementary actions, any subgoal is defined by merging it with the physical role and the concerned component. A small number of pre-defined generic verbs Dapoigny, Benoit, & Foulloy (2003), creates elementary sub-goals which can be joined to form more complex goals at the service level or the mode level. An example of functional mereology is detailed in subsection *Application*.

Behavioral knowledge

In related areas such as dynamic logic, actions are described through preconditions and effects Harel (1984). Preconditions capture the fact that actions cannot be executable in all states. Similarly, we define the behavior as based on pre-(post)conditions and actions which must represent the available processes in the instruments. The behavioral theory reflects the causality of the system with a representation by causal predicates. In order to relate functional concepts to the behavioral model, we argue that a given behavior can be derived from two root concepts, the event concept (that can be seen as a predicate true if this event has occurred) and the action concept. As the action concept is also related to the functional mereology, the functional mereology will be extended by supplementary concepts to form the behavioral mereo-topology. A similar result is pointed out in Kitamura & Mizoguchi (1998) which argue that the function is inseparable from the behavior. Events establish a causal link between atomics behaviors.

Definition 3 *An event which represents an asynchronous signal between processes is formalized by the predicate $Achieved(a, p)$ which becomes true as soon as the action a is complete in process p.*

Definition 4 *An action represents the execution of a given (meta)function conditioned with input conditions (pre-conditions).*

While a goal and its associated action have similar representations, the goal is an intensional concept whereas the action denotes the realization of this goal. The instrument model must include a set of predicates and relations are meaningful for further elaboration of the behavioral specification. As mentioned in Kmenta, Fitch, & Ishii (1999), a behavior can be decomposed into smaller atomic behaviors. We define an elementary or atomic behavior and a complex behavior as follows:

Definition 5 *An atomic behavior b_{ij} is represented by the set :*

$$b_{ij} = \{\Pi^I_{ij} \cup Exec(a_j) \cup \Pi^O_{ij}\}, \quad \Pi^I_{ij} \in \mathcal{P}^n, \Pi^O_{ij} \in \mathcal{P}^m, \text{ and } a_j \in \mathcal{A} \qquad (2)$$

where $\mathcal{A}$ and $\mathcal{P}$ are the sets associated with the respective sorts of elementary actions and the sort of time-dependent predicates also known as fluents, n and m the respective numbers of input and output predicates, I and O denoting the respective input and output predicates.

Definition 6 *A complex behavior is seen as a causally ordered sequence of atomic behaviors that satisfies a goal.*

The topology relies on the connect primitive, where the connection is related to the input conditions also called input predicates vector. Input conditions have a mandatory element, the input event (i.e., $P_0 = Occurred(e_i)$), and optional input predicates. Any service describes a behavior expressed by a graph where the functional nodes are represented with a single goal and available actions selected accordingly with pre-conditions. Goals and sub-goals are related through the function mereology while actions are selected with input conditions and topologically connected with event occurrences.

Application

Consider for example, a color sensor which is based on the spectrum measurement of light. Four modes are implemented, one for the configuration, and three for measurements. The xyz, $YC1C2$ and Lab modes specifies the users

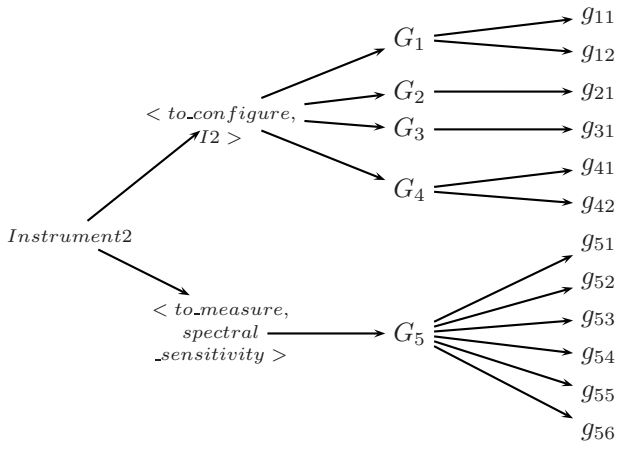

Figure 2: The color sensor functional mereology.

choice for chromaticity coordinates. As their internal structure is quite similar, we restrict the discussion to the xyz mode. Internal events are either user-defined or available in libraries (in this case there are hardware-dependent).

$G_1 = < to\_configure, whiteBalance, sample >$
$G_2 = < to\_configure, spectrumPrecision, sample >$
$G_3 = < to\_configure, acquisitionDuration, sample >$
$G_4 = < to\_configure, defaultSpectrumPrecision, sample, defaultAcquisitionDuration, sample >$
$G_5 = < to\_measure, spectrum, Sample >$
$g_{11} = < to\_acquire, rawSpectrum, sample >,$
$g_{12} = < to\_compute, XwhiteSpectralSensitivity, sample, YwhiteSpectralSensitivity, sample, ZwhiteSpectralSensitivity, sample >,$
$g_{21} = < to\_receive, measurementPrecision, sample >,$
$g_{31} = < to\_receive, exposureDuration, sample >,$
$g_{41} = < to\_initalize, XwhiteSpectralSensitivity, sample, YwhiteSpectralSensitivity, sample, ZwhiteSpectralSensitivity, sample >,$
$g_{42} = < to\_initialize, exposureDuration, sample >$
$g_{51} = < to\_receive, G_5, dest >$
$g_{52} = < to\_acquire, raw_spectrum, area >$
$g_{53} = < to\_compute, XSpectralSensitivity, area, YSpectralSensitivity, area, ZSpectralSensitivity, area >$
$g_{54} = < to\_compute, FuzzyColor, Sample >$
$g_{55} = < to\_compute, spectrum, area >$
$g_{56} = < to\_send, spectrum, Sample >$
$g_{57} = < to\_send, < ext\_goal >, dest >$

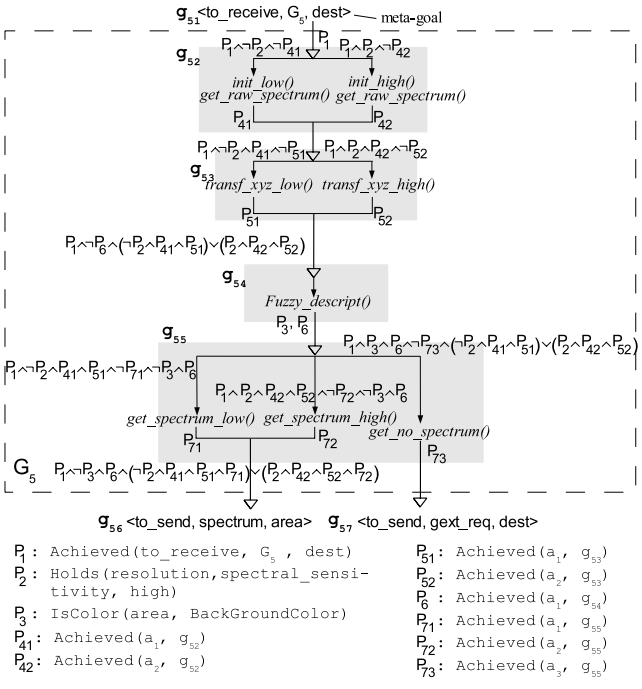

Figure 3: The color sensor behavioral mereo-topology.

The Configuration mode holds first a service dedicated to the adjustment of the white color related to a goal G_1, a goal G_2 defining the desired precision for spectrum measurement, a goal G_3 which specifies user requirements for the spectrum acquisition duration and finally, a goal G_4 allowing default

initializations. In the xyz measurement mode, a single service is related to the global goal G_5. Each service goal is related with one or more sub-goals as described below. The corresponding mereology is reported in figure 2. Alternatively, figure 3 presents a behavior mereo-topology (in the measurement mode) related to goal G_5 and the possible pathes (i.e., possible behaviors) in the topology. Initially, all $Achieved()$-type predicates are false and the process starts as soon as P_1 is true. Two different behaviors are related to the desired resolution selected in the configuration mode where G_2 is available, whereas an alternative behavior can occur if the area color appears to be that of the background (predicate P_3 true). In that last case, the instrument sends an external goal request to a dependent actuator in order to rotate the sample holder.

Implementation

The Java-based software Graphical Captool is dedicated to II design. It relies on a top-down approach with XML as programming language. The software architecture provides five layers interacting with four libraries which are respectively:

- GCSysEditio intended for graphical modelling.
- GCServiceEditio dedicated to service modelling.
- GCXML XML, RDF and DAML for flow management.
- GCObComp for source codes generation.

As already mentioned, the semantic structure design of II is based both on a physical ontology (PO) and a behavioral ontology (BO). A functional description is provided, either in the C or Java languages. A layout of the graphical architecture of the design tool is depicted in figure 4. In a first step, the user must create a system including instruments attached to a single PO. In a second step, he describes the physical environment (which generates the PO) and finally, the available behaviors (BO). In such a way, a simulation platform for II can be investigated. As a consequence, any instrument can be exported to another system, its proper required ontology being merged with the destination one. The Instrument modelling is composed with proper parts ($PP(xy) \stackrel{def}{\equiv} P(xy) \wedge \neg x = y$, e.g. $PP(mode, ii)$) which are reusable. They are stored in libraries and can be duplicated or aggregated for template use. Templates are saved as XML files according to an RDF description. The programming language for II is supplied by the RDF meta-model. The system meta-model is composed of mereological axioms where the XML XPointers provide links between separate $proper\_part\_ofs$. The model describing objects which are part of the software architecture follows the design pattern MVC (Model View Component) implemented by Swing components. In such a way, the system components can be seen either as a metaphorical representation, or as a RDF description of templates. Therefore, there exists a semiotic morphism Malcom & Goguen (1998)Goguen (1999) between the graphical model and RDF description

$$\begin{aligned} &\exists \mathcal{R} : (Graph \rightarrow RDF)(* \in Graph \wedge (o \in RDF)) \\ &such\ as\ \mathcal{R}(a * b) = \mathcal{R}(a) o \mathcal{R}(b))\ a, b \in Graph \\ &i.e.\ Graph\ \cong\ RDF \end{aligned} \tag{3}$$

these two models follow the same composition rules (e.g. the mereological rules extracted from the functional model). The analysis of the graphical models is achieved with a parser. The resulting meta-model is compiled to provide a Java agent. The XML file is converted into an object whose methods can be reified (with Prolog rules.) The Java programming language has been selected because it offers a number of significant advantages. It is the appropriate language for Web applications with a wide range of libraries for graph handling and XML files. Graph management relies on two libraries, JGraph which allows specific graph design (e.g., for II mode description), and ZVTM from Xerox (grounding for RDF browser of W3C), appropriate for complex graphs. In order to preserve inter-operability, we adopt meta data standard for output files (i.e., XML format). XML file management requires both the Xerces (XML) and Jena libraries (for RDF and DAML descriptions). The mereological and topological descriptions extend the RDF semantics with Prolog rules as suggested in Conen & Klapsing (2002). Service descriptions take benefits from the $DAML-S$ description language with the restriction to a subset ($DAML-S\alpha\iota$). Services which can be either compound services, or atomic services are merged inside a dedicated library.

Graphical behavior design. Each model is parsed through a Java-based inference engine involving the XProlog library Kalfoglou & Robertson (1999). For instance, during the design step, one can find (via ontologies) a given service knowing its goal, sub-goal or effects. Internal services operating sequence (atomic services) which is C- or Java-implemented is captured inside the Model part of $DAML-S\alpha\iota$ service description.

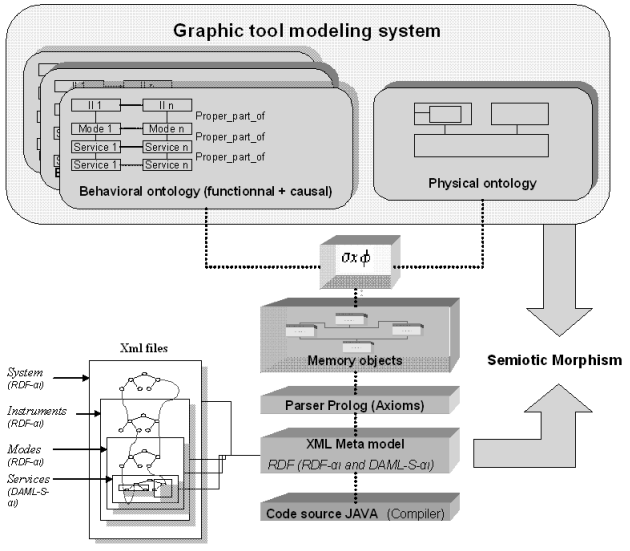

Figure 4: Overview of the GUI architecture

Conclusion

The approach described in this paper proposes an automatic construction of a knowledge base which is appropriate for sensor/actuators control. Significant contribution appears first on the mereo-topological approach which provides more accurate structuring of knowledge allowing for easy formal modelling and multi-agent implementation. Another contribution comes from the GUI architecture which captures appropriate information from behavioral requirements input by user. He only describes physical, functional and behavioral constraints for the system while the GUI constructs (both from users and libraries information) knowledge structures which are consistent and which will serve as a basis for reasoning and distributed planning in the operational phase. Further works concern a high-level programming language suitable for dynamic planning centered on reified techniques Douence & Sudholt (2001) with an XML description. Alternative works will investigate agent implementation on II to enable automatic remote service integration based on data dependencies through the network of II.

References

Benoit, E.; Foulloy, L.; and Tailland, J. 2001. Inoms model: a service-based approach to intelligent instrument design. In *Proceedings of the fifth Conference on Systemics, Cybernetics and Informatics*, 160–164.

Borst, P.; Akkermans, H.; Pos, A.; and Top, J. 1995. The PhysSys ontology for physical systems. In *9th international Workshop on Qualitative Reasoning*, 11–21.

Bouras, A., and Staroswiecki, M. 1997. How can intelligent instruments interoperate in an application framework? a mechanism for taking into account operating constraints. In *IFAC Symposium on Intelligent Components and Instruments for Control Applications (SICICA)*, 465–472.

Bruegge, B., and Dutoit, A. 1999. *Object-Oriented Software Engineering; Conquering Complex and Changing Systems*. Prentice Hall PTR.

Conen, W., and Klapsing, R. 2002. Utilizing host formalisms to extend RDF semantics. In *Proceedings of German Wirtschaftsinformatik Conference*.

Dapoigny, R.; Benoit, E.; and Foulloy, L. 2003. Functional Ontology for Intelligent Instruments. In Zhong, N.; Ras, Z.; Tsumoto, S.; and Suzuki, E., eds., *Foundations of Intelligent Systems*, number 2871 in LNAI, 88–92. Springer.

Dooley, K.; Skilton, P.; and Anderson, J. 1998. Process knowledge bases: Facilitating reasoning through cause and effect thinking. *Human Systems Management* 17(4):281–298.

Douence, R., and Sudholt, M. 2001. A generic reification technique for object-oriented reflective languages. *Higher-order and Symbolic Computation* 14(1):7–34.

Goguen, J. 1999. An introduction to algebraic semiotics, with applications to user interface design. In Nehaniv, C., ed., *Computation for metaphors, analogy and agents*, volume 1562 of *LNAI*, 242–291. Springer.

Gruber, G., and Olsen, G. 1994. An ontology for engineering mathematics. In Doyle, J.; Torasso, P.; and Sandewall, E., eds., *Fourth International Conference on Principles of Knowledge Representation and Reasoning*, 258–269. Morgan Kaufmann.

Harel, D. 1984. Dynamic logic. *Handbook of Philosophical Logic* 2:497–604.

Hertzberg, J., and Thiebaux, S. 1994. Turning an action formalism into a planner: a case study. *Journal of Logic and Computation* 4:617–654.

Kalfoglou, Y., and Robertson, D. 1999. Managing ontological constraints. In *Proceedings of IJCAI'99*. AAAI Press.

Kitamura, Y., and Mizoguchi, R. 1998. Functional ontology for functional understanding. In *Procs. of the 12th International Workshop on Qualitative Reasoning*, 77–87. AAAI Press.

Kmenta, S.; Fitch, P.; and Ishii, K. 1999. Advanced failure modes and effects analysis of complex processes. In *Procs. of the ASME Design Engineering Technical Conferences*, number DETC99/DFM-8939.

Lifschitz, V. 1993. A theory of actions. In Kaufmann, M., ed., *Proceedings of the tenth International Joint Conference on Artificial Intelligence*, 432–437.

Lind, M. 1994. Modeling goals and functions of complex industrial plant. *Journal of Applied Artificial Intelligence* 8:259–283.

Malcom, G., and Goguen, J. 1998. Signs and representations: Semiotics for user interface design. In Paton, R., and Neilson, I., eds., *Visual Representations and Interpretations*, Springer Workshops in Computing. Springer.

Miles, L. 1981. *Techniques of Value Analysis and Engineering*. Mac Graw Hill.

Salustri, F. 1998. Function modeling for an integrated framework : A progress report. In Cook, D., ed., *Procs. of FLAIRS98*, 339–343. AAAI Press.

Simons, P. 1987. *Parts: a Study in Ontology*. Oxford University Press.

Smith, B. 1996. Mereotopology: A theory of parts and boundaries. *Data and Knowledge Engineering* 20:287–303.

Spalazzi, L., and Traverso, P. 1999. A dynamic logic for acting, sensing and planning. *J. of Logic Computation* 9(97-73):1–36. Oxford University Press.

Toppano, E. 2002. Mmforted: a cognitive fostering the acquisition of conceptual knowledge about design products. *Artificial Intelligence in design'02* 191–214.

Umeda, Y., and al. 1996. Supporting conceptual design based on the function-behavior-state modeler. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 10(4):275–288.

Varzi, A. 1996. Parts, whole and part-whole relations: the prospect of mereotopology. *Data and Knowledge Engineering* 20:259–286.